DISK BASIC MANUAL

MA-S101 .

FOR

HITACHI PERSONAL COMPUTER

MB-S1

This manual is a translation of the manual for Personal computer MB-S1 10/20 for use in Japan.
Some of the specifications may be changed in export models.

DISK BASIC MA-S101

PREFACE

This instruction manual describes DISK BASIC MA-S101 of the Hitachi MA-S1 Series personal computers.
On the DISK BASIC, instructions and functions are added to operate the floppy disk for the S1 BASIC mounted to MB-S1. When compared with the previous DISK BASIC (MA-5310/MA-5320) for Basic Master Level 3, DISK BASIC MA-S101 has such features as higher version compatibility in the language specifications and file format, ability of using the standard floppy disk and RAM disk in addition to a mini-floppy disk, and additional statements like CHAIN and COMMON. This manual explains these functions added to the S1 BASIC.

For details of the S1 BASIC, refer to the "BASIC manual" of MB-S1 and for detailes of the hardware, refer to the instruction manuals of MB-S1 and individual devices.

1. Operation Procedures

1.1 How to read this manual

This instruction manual is prepared for DISK BASIC MA-S101 (hereinafter referred to as DISK BASIC) for the Hitachi MB-S1 series personal computers (hereinafter referred to as S1). It consists of six independent chapters, and these

chapters contain the following explanation.

Chapter 1. Operation Procedures

This chapter explains the basic points including the connecting method of an S1 unit with a mini-floppy disk (MP-3560) and starting method of the DISK BASIC and how to handle the floppy disk.

Chapter 2. Outline of DISK BASIC

This chapter describes the concept and types of files and floppy disk.

Chapter 3. Utilities

This chapter describes the DISK BASIC utilities.

Chapter 4. Language Specifications

This chapter describes the DISK BASIC instructions and functions.

4.1 Instructions: This clause shows all commands and statements related to the DISK BASIC disk operation in the alphabetic sequence, as well as describing their functions and formats.

4.2 Functions: This clause describes the functions related to the DISK BASIC disk operation in the following sequence.

PURPOSE : Shows what the instruction or function does.

FORMAT : Shows how to write parameters and their sequence on the instruction or function. The meaning of each sign is as follows:

1. Items written in capital letters of alphabet must be input as is.
2. The user specifies the items enclosed in ⌜< >⌟.
3. The specification of items enclosed in brackets ⌜[ ]⌟ is optional and it may be omitted. If the specification is omitted, a default value or a value specified beforehand is used depending on the instruction or function.
4. Items where the omission sign of ⌜...⌟ is written can be specified as many times as needed within the limit of one line. For example, if a format is shown as ⌜<variable> [,< variable>] ...⌟ , it can be written like ⌜A, B, X, Y⌟.
5. A format description of ⌜;,⌟ means either the top one or bottom one can be used. In this case, either a semi-colon ⌜;⌟ or a comma ⌜,⌟ can be used.
6. Signs other than the above, including a comma and period ⌜.⌟ for omission, are signs that are essential on each syntax. All these signs, except signs that can be omitted singly (for example ⌜,⌟ when ⌜[,]⌟ is given), must be described. (For example, if ⌜[,R]⌟ is given, when ⌜[R]⌟ is omitted, ⌜[,]⌟ can also be omitted, but when ⌜[R]⌟ is written ⌜[,]⌟ must also be written.)

7. In the explanation, ⌜ ⌟ are used as quotation marks. The quotation mark of ⌜"⌟ means a sign used in programs.

EXPLANTION : This section explains actual actien or value of the instruction or function and cautionary points on use.

Chapter 5. File Operation

This chapter explains the operation method of disk files and writing methods of commands and programs using actual examples. The explanation is given assuming that the reader has mastered the knowledge on the BASIC (hereinafter referred to as S1 BASIC) of the MB-S1 series personal computers, but the disk explanation is given in a way that beginners can easily understand. We recommend that beginners read this part first.

Chapter 6 Use Method of RAM Disk

This chapter explains how to use the RAM disk.

Chapter 7 Reference Materials

ASCII Code Table and error messages are contained in this chapter.

After reading this manual and before starting to use the DISK BASIC, users are requested to read the S1 BASIC manual and instruction manual of Mini-floppy Disk MP-3560 and fully

understand the S1 BASIC specifications and connection method of MP-3560 and S1. Also, we recommend that readers who use the DISK BASIC for the first time read this manual in the sequence of Chapter 1, Chapter 5 and Chapter 2.

The description of this manual is different from that of the S1 BASIC Manual on the following points:

1. Chapter 4 describes instructions and functions related to the disk only. For explanation of instructions and functions that are not related with the disk, refer to the S1 BASIC Manual.
2. 「 」 are used as quotation marks in all explanatory sentenses, distinguishing from 「" "」 used as syntax.

1.2 Starting of DISK BASIC

The following hardware configuration is necessary to use this DISK BASIC:

. Hitachi personal computer S1	MB-S1 10/20
. Mini-floppy disk	MP-3560 (or MP-3550)
. Mini-floppy disk card	MP-1870 (or MP-1802)
. Display	Refer to the S1 instruction manual.

Read Chapter 3, Connection of Main Unit and Periperal Devices on the S1 Instruction Manual and instruction manuals of related devices first. Connect an S1 unit which has been

extended for an additional mini-floppy card with a mini-floppy disk and a display unit and start the DISK BASIC in the following procedures.

(1) Turn on the power switch of the mini-floppy disk and display unit.

(2) Mount a system diskette provided onto Drive 0 of the mini-floppy disk and turn on the power switch of S1. When the DISK BASIC is started and the DISK BASIC is set to the operating state on S1, the following message is output to the screen.

×××

(3) The 「24K bytes」 shown on the message fourth line indicates the number of bytes of memory that can be used for the program. This is a standard value and it varies when a NEW ON instruction is executed, extension state of the RAM, and the value specified for the utility of 「Setting of system attribute」 .
When restarting the DISK BASIC after executing a NEW ON instruction or resetting, the system diskette must be mounted to Drive 0 of the mini-floppy disk.

(4) When the power switch of mini-floppy disk is turned on, if Drive 0 is not mounted with the system diskette or if it is not mounted properly, a message of 「Insert Program Diskette」 is displayed at the screen center and the system waits for mounting of the system diskette.

The DISK BASIC starts as soon as the system diskette is mounted correctly.

The following causes are conceivable for incorrect display on the screen. Check these in such a case.

(a) The power cord for S1, mini-floppy disk or display unit is not plugged in.

(b) The mini-floppy disk connector is not correctly connected with the mini-floppy disk card.

(c) The system diskette is not correctly mounted to Drive 0 of the mini-floppy disk.

(d) The power for S1 and mini-floppy disk was not turned on in the correct sequence. (The power for mini-floppy disk should be turned on first.)

1.3 Handling of floppy disk (diskette)

1.3.1 Preparation of backup diskette for system diskette

In order to avoid destruction of the provided system diskette by an erroneous operation. a backup diskette must be made in advance. Make a copy of the system diskette in the following procedures.

① Stick a write protect seal to the write enable notch of system diskette, so that writing to it is prohibited.

② Prepare a new diskette or an old diskette on which the recorded program or data is no longer needed.

③ Mount the system diskette to Drive 0 and the diskette

for copying to Drive 1, and initialize the diskette following the explanation given in 「1.3.2 Diskette initialization」.

④ When the initialization terminates and 「N」 is input to the message of 「Continued (Y/N)?」, the display should be on the submenu of 「Format, disk copy」. Key in 「2」 at this stage.

⑤ The following message is displayed.

××××

⑥ Check that the system diskette is mounted to Drive 0 and the initialized diskette is mounted to Drive 1 once again, and key in 「Y」. If 「N」 is keyed in, the display returns to the submenu.

⑦ When the disk copying normally terminates, the following message is output.

⑧ Key in 「Y」 to continue the disk copying. Key in 「Y」 to terminate the copying.

⑨ Reserve the provided system diskette and use the copy diskette on daily operation.

1.3.2 Diskette initialization

A new diskette must be initialized before it is used. The follwing shows the initialization procedures.

① First, mount the system diskette to Drive 0 and restart the DISK BASIC by applying 「NEW ON 1」 instruction.

② Then, key is as follows:

RUN "UTILITY"

③ The main menu as shown below is displayed. Key in Utility No. 「1」 (Utility No. 「2」 to initialize the standard floppy disk.)

④ The follwowing submenu is displayed. (When Utility No. 2 is input, 「5 inches」 is displayed instead of 「8 inches」.) Key in 「1」.

⑤ The following message is displayed.

Mount a diskette to initialize (a new diskette or old diskette on which the recorded program or data is no longer needed) to an empty drive among Drives 1 to 3, and key in the Drive No.

⑥ The following message is displayed.

Check once again that the mounted diskette is the diskette to initialize and key in 「Y」. If 「N」 is keyed in, the display returns to a submenu and no initialization is conducted.

⑦ Keying in of 「Y」 starts fomatting and initialization. When the processing terminates, the following mesaage is diaplyed.

Key in 「Y」 to continue the formatting. Key in 「N」 to

terminate the processing.

When initializing again an old diskette which has been initialized (formatted and initialized) and used, the formatting is unncecessary. Use a DSK INI instruction for the initialization.

Note that since a standard floppy disk has the standard format, the initialization can be conducted by a DISK INI instruction only. However, if this is done, there may be cases of unable to read Sectors 9 through 52 of Cylinder 0 by a DSKI$ statement or unable to dump them by the utility of ⌜Disk Dump⌟. Since the DISK BASIC does not use these sectors, this causes no problem on normal operation. However, if this creates a problem, the formatting must be redone.

(Refer to Chapter 4 for explanation of DSK INI instruction.)

1.3.3 Caution on handling

Pay attention to the following points when handling a floppy disk (diskette).

(1) Do not apply unnecessary force to the diskette. Never bend the diskette, apply a clip, nor place anything that is heavy on it.

(2) Do not directly touch the recording face of diskette with bare hand. Pay attention to this caution when putting in or taking out a diskette from the envelope.

Be careful not to stain the diskette.

(3) Make sure to keep the diskette in the envelope when it is not used. When storing it, place the envelope containing a diskette in a case and store it in a way that no pressure is applied to it. Also, keep the empty envelope in the storage case.

(4) Do not clean the diskette.

(5) Do not expose the diskette to the direct sunshine or high temperature (50°C or above). When using a diskette that has been stored at a temperature lower than the operating temperature of the device, first adapt itself to the use environment while keeping it in the envelope. The temperature gradient is 20°C/hour.

(6) Do not place a magnet or anything that has magnetism near the diskette. When the diskette is exposed to a magnetic field of 50 Oe or greater, the recorded data may be destroyed or errors may occur.

(7) Do not write on the diskette directly. Use a predetermined label always. Write on this label and stick it to the diskette.

Use a felt pen when writing. Never use a ball-point pen, pencil or eraser. (The powder from eraser sticks or damage the diskette surface, resulting in a trouble.)

(8) When writing on a label that is pasted on the diskette,

the diskette must be kept in the envelope and the envelope must be placed on a flat surface. Write on the label while paying attention not to let the envelope touch the recorded surface.

(9) When replacing a new label, peel off the old label, remove the adhesive and then stick the new label. If a new label is pasted on top of the old label, the excessive label thickness causes a trouble to the operation.

(10) Gently insert the diskette straightly until it reaches the end. Inserting it diagonally or violently damages the diskette, making it unusable.

(11) Do not remove the diskette when the lamp of the drive is lighting to show that it is being acessed. If this is done, the diskette or drive head may be damaged, making them unusable.

(12) If execution of a program that uses a data file is temporarily stopped for such a reason as an error has occurred, [BREAK], [CTRL] + [C], [CTRL] + [D] or [BREAK] + [RESET] was input, or a STOP or ERROR instruction was executed, the file that was opened by an OPEN instruction is kept in the open state. Therefore, a CLOSE statement must be executed without fail except the case of restarting the execution by a CONT instruction

or the like. If a CLOSE instruction is not executed, the following case occurs.

(a) When a file is renewed, unless a CLOSE instruction is executed, the contents of diskette containing the renewed file are not guaranteed.

(b) If the diskette is exchanged without closing the file in Case (a) of the above, the exchanged diskette is destroyed when a CLOSE instruction is given later. When it is destructed, the number of groups displayed by a FILES instruction becomes erroneous or an error called 「Bad File Structure」 results in. Once destroyed, the diskette cannot be recovered. Use it by initializing it by a DSKINI instruction.

(13) Pay attention to the following points when an error has occurred during execution of an instruction that uses a disk, or temporarily stopping the execution by inputting [BREAK] + [RESET] or [CTRL] + [D].

(a) If [BREAK] + [RESET] or [CTRL] + [D] is input in the following state, since the execution is forcibly stopping without completing the processing, the disk may be destroyed or various discrepancies may occur.

(i) During execution of a KILL, NAME or DSKINI instruction

(ii) During disk renewal for open processing by an OPEN ("O" mode), SAVE or SAVEM instruction

(iii) During disk renewal for open processing in the case of preparing a new file by an OPEN ("R" mode) instruction.

(iv) During disk renewal for close processing by a SAVE, SAVEM, CLOSE or END instruction or for the subsequent processing after inputting |BREAK| + |RESET| or |CTRL| + |D| .

(b) If an error occurs during execution of the following instructions, no subsequent processing (close or other processing) is not conducted. Press |CTRL| + |D|. The subsequent processing is conducted. However, in this case, the execution cannot be restarted by a CONT instruction.

LOAD	SAVE
LOADM	SAVEM
MERGE	LIST
CHAIN	RUN

(14) When |BREAK| + |RESET| or |CTRL| + |D| is input, the processing is forcibly cancelled. Therefore, discrepancies occur in the processing after that.

The following shows examples of these discrepancies.

(a) When one of the above key input is made during text editing (including RENUM, MERGE, CHAIN MERGE), the text may be destroyed. Reload.

(b) When one of the above key input is made during calculation or assignment processing to a variable, the contents of variable or array are not guaranteeed. Therefore, to stop the execution temporarily, press [BREAK] (or [CTRL] + [C]) first. Press [CTRL] + [D] only when the above did not stop the execution. Pressing [BREAK] + [RESET] creates a higher probability of generating discrepancies. Use of this input must be limited to the case of [CTRL] + [D] input is not effective.
This caution applies samely to the DISK BASIC and S1 BASIC.

(15) If one of the following applies to a diskette, do not use the diskette but replace it with a new one.

- If a diskette is broken, distorted, bent, partially tipped off, the edge is crushed, or any other physical damage has been given.
- If the recorded surface or jacket is fouled by sticky liquid like refreshment drink or coffee, solvent or metal powder.
- If the diskette recording face is damaged or stained.

If a diskette having any one of these conditions, it may damage the head or drive, causing a trouble.
Also note that if the head stained or damaged by one diskette is used on a different diskette, the second

diskette may also be damaged.

(16) Normal write or read is unavailable if the following processing is conducted. Yet, since it does not create an error, users must pay close attention to it.
(No direct error is created, but other errors may be generated indirectly.)

(a) Applying OPEN to an existing sequential file in the "R" mode.

(b) Applying OPEN to an existing random file in the "I" mode.

(c) Applying OPEN to an existing file in the "O" mode (the existing file is killed).

(d) Applying OPEN to an existing random file using a record length that is different from the length given to it initially.

(e) Allocating a data formation (FIELD statement) that is different from the structure given to it initially to an existing random file.

If write is attempted in the cases of (a), (d) or (e), even other programs which have been operating normally may become abnormal.

In the case of (d), even the LOF function does not give a correct numeric. Therefore, when using a file, control the use by discriminating the file with a

file extender name or the like in order to avoid this type error.

(17) Apply the write protect function to any diskette, like the system diskette, on which the contents should not be rewritten by an error. When write protect is applied, write to the floppy disk is disabled and this will avoid destruction of diskettes by an error as described for cases (12) through (16) in the above. The method of applying the write protect function varies by the difference of mini-floppy disk and standard floppy disk. For details, refer to the instruction manual for the floppy disk.

11

2. Outline of DISK BASIC

2.1 File

2.1.1 Concept of file

The concept of 「file」 is used by the S1 BASIC when recording or writing a program or data to an auxiliary memory medium (floppy disk, cassette tape) or when inputting or outputting data with an input device (keyboard, display).

Think of a file that is used for arrangement or storage of documents. Men classify documents or tables and figures that are useful later, file them for each classification, and each file is given the title and stored on shelves. When

certain information in the file is needed, a file that contains the information is searched based on the title of file, and the file is opened and the necessary information is read. Also, the file system is very useful when certain information is transferred from a person to another person in a form of document, table, or figure.

Let's apply this file image to the S1 BASIC processing. The information referred to in the above is a program or data. Imagine that one program or a group of data is stored in a file. Each 「file」 is given a 「file name」 that is equivalent to the title referred to in the above. The place to store the file, that is, a shelf in the above explanation, is an auxiliary memory medium like a floppy disk.
Thus, all programs and data are stored, input or output in the form of 「file」 by the S1 BASIC.

2.1.2 File descriptor

Since all input and output are processed in the file system, input/output by the S1 BASIC can be executed by the same instruction even if input/output devices are different. The device to handle the input/output processing is specified by a 「file descriptor」, and the objective file (device) is switched by the software.
A 「file descriptor」 is written in the following format.

"[<device name>:][(<option>)][<file name>[.<file name descriptor>]]"

12

1. Device 2. Device name 3. Input
4. Output 5. Remarks 6. Keyboard
7. Screen 8. Printer
9. Needs extended printer card MP-1810. 10. RS-232C port
11. Needs RS-232C connector MP-9732.
12. Needs extended RS-232C card MP-1802. 13. Cassette recorder
14. Deault of ROM BASIC 15. Disk drive
16. Default of DISK BASIC 17. Needs MP-1801 when MP-3370 is used.
18. Needs MP-1801 when MP-3370/3375 is used.

<device name> : A device name is given to each input/output device. On the DISK BASIC, the <device name> is defined as shown in the table above. When the specification of <device name> is omitted, Drive 0 ("0:") of the floppy disk unit is assumed.

<option> : This must be specified when the RS-232C line is used. For detailed explanation, refer to 「BASIC Manual」 and 「Instruction Manual」 of S1.

<file name> : Since the floppy disk and cassette recorder can contain a number of files in the same input/output device, the <file name> must be

specified. This specification is given in a string consisting of no more than eight characters and symbols excluding 「(」, 「)」, 「.」 , and 「:」 .

file name extender: This accomapnies a file name and it may be considered as a part of file name. It is described in up to three characters immediately after 「.」 . The characters usable for a file name can also be used for this. When the <file name extender> specification is omitted, 「.BAS」 is given for LOAD, SAVE, LIST, MERGE and RUN instructions, 「.BIN」 for LOADM and SAVEM statements, 「.DAT」 for OPEN statement, and 「. ␣ ␣ ␣」(␣ is a space) for NAME and KILL instructions.

2.1.3 Types of files

Files can be classified from various viewpoints, for example, whether the file is a program or data. On the DISK BASIC, files are classified as follows by the difference of file preparation methods.

1. File types 2. Item No. 3. File creation method
4. File classification 5. Data type 6. Data form
7. Access method 8. File name extender given
9. Indication by FILES instruction 10. BASIC program

11. Internal expression (binary) 12. Sequential

13. Character (ASCII) 14. Machine language program

15. Data 16. Random

A data type means a classified type of data in files. A file made by a SAVE instruction is regarded as a BASIC program file, a file made by a SAVEM statement is regarded as a machine language program, and a file made by an OPEN statement is regarded as a data file.

The data form is classified by the difference of form (expression) in which the data is described. A form in which data on a memory is recorded in a file as is (as expressed internally) is the binary form. If the internal expression of data is converted into character strings and recorded in a file, it is the ASCII form. This classification is very important with BASIC program files, and there are the following differences.

17. Comparison of binary and ASCII forms 18. Comparison item

19. Binary form 20. ASCII form

21. A program saved by L3 BASIC is loaded by S1 BASIC, or vice versa.

22. Impossible 23. Possible

24. Use a MERGE or CHAIN MERGE instruction.

25. A saved program is read and processed by the BSSIC using a LINE INPUT# statement or others.

26. Difficult 27. Easy

28. A file (Note) output by the BASIC using a PRINT# statement

or others is loaded as a program file.

29. Speed of LOAD instruction 30. Fast

31. Slow 32. Use ratio of disk 33. Good

34. Not good 35. (Note) Must be in the correct program form.

There is another method of classifying files; sequential file and random file, which is made by the difference of access method. Sequential files are continuously recorded as on a recording tape and the contents are read in the same sequence as they were recorded. On random files, each data item is given a record No. and an objective data item can be directly accessed (input/output) regardless of the recording sequence. For further details, refer to Chapter 5. File Operation. However, on this DISK BASIC, files on a disk are prepared in the same structure regardless of the difference of a sequential or random file and there is no distinction. Therefore, the same file can be used either as a sequential file or as a random file, which is convenient if it is used correctly. Even if it is not used correctly, no error is displayed, and users must be careful on this point.

Among various classification methods, the data type and data form are controlled by the DISK BASIC. Therefore, they can be checked using a FILES instruction. For details, refer to the exaplanation of FILES instruction in Chapter 4. Language Specifications.

2.2 Floppy disk

A new floppy disk must be initialized. Initialize it following the explanation given in 1.3.2 Diskette initialization. To use a diskette that has been formatted in other method or when data is destroyed on a disket and it cannot be read, initialize such a disk. However, a damaged diskette cannot be used again even after initialization. The system diskette needs no initialization. Also, apply the write protect function to the system diskette provided by us to avoid its contents destroyed by an error. For details of applying the write protect function, refer to the floppy disk instruction manual.

The following explains the recording mechanism of diskette.

2.2.1 Mini-floppy disk

Data is recorded on a diskette by magnetism in the concentric form as shown below.

1. Write enable notch 2. Track 3. Diskette
4. Sector 5. Side

Each concentric circle on one side (Side 0 or Side 1) is called a track. A group of corresponding tracks on Side 1 is called a cylinder. One diskette consists of 40 cylinders numbered 0 to 39, and each cylinder is divided into 32 sectors numbered 1 to 32. Sectors 1 through 16 are on Side 0 and sectors 17 through 32 are on

Side 1. One sector consists of 256 bytes.

This DISK BASIC controls the area on disk in the unit called a group. On a mini-floppy disk, one group consists of eight sectors, and 38 cylinders out of cylinders 1 through 39 excluding Cylinder 20 are divided into 152 groups consisting of Group 0 through Group 152 and are used as areas that can be written into or read out of.

A diskette is configurated as follows.

Cylinder 20 contains FAT (File Allocation Table) and a directory, and this cylinder is called a file control cylinder. For further explanation, refer to 2.2.4 File control method.

16 1. Cylinder 2. Sector 3. Side 0

4. Side 1 5. BL: Bootstrap loader

6. L: DISK BASIC loader (Note) 7. S: System code area (Note)

8. U: User area 9. N: Unused 10. D: Directory

11. (Note) These are user areas on a diskette if it not a system diskette.

2.2.2 Standard floppy disk

Data is recorded on a disket by magnetism in the concentric form as shown below.

12. Track (数字) 13. Diskette 14. Sector (数字)

17 Each concentric circle on one side (Side 0 or Side 1) is

called a track, and the corresponding tracks on Side 0 and Side 1 are collectively called a cylinder. One diskette has 77 cylinders numbered 0 to 76, and each cylinder is divided into sectors 1 through 52. Sectors 1 through 26 are on Side 0 and sectors 27 through 52 are on Side 1. One sector consists of 256 bytes. However, Cylinder 0 at Side 0 is recorded in single precision and and here one sector consists of 128 bytes.

This DISK BASIC controls the area on disk in the unit called a group. One group consists of 26 sectors, and 73 cylinders out of cylinders 1 through 74 excluding Cylinder 37 are divided into 146 groups of 0 through 145 and are used as areas that can be written into or read out of.

Cylinder 37 contains FAT (File Allocation Table) and a directory, and this cylinder is called a file control cylinder. For further explanation, refer to 2.2.4 File control method.

A diskette is configurated as follows.

1. Side 0 2. Side 1
3. BL: Bootstrap loader L: DISK BASIC loader (Note)
   S: System code (Note) U: User area
   F: FAT D: Directory N: Unused
4. (Note) These are user areas on a diskette if it is not a system diskette.

2.2.3 RAM disk

This DISK BASIC can use a RAM disk. For explanation of how to use it, refer to 6. Use Method of RAM Disk.

Here, the RAM disk recording structure is explained.

A RAM disk is used to provide faster input/output of data using a memory (RAM) on S1 instead of a diskette as recording medium. However, being RAM, it has the shortcoming of the recorded data being erased when the power is turned off. These characteristics must be kept in mind when a RAM disk is used.

A RAM disk is initialized at the time the DISK BASIC is started, and at this stage, Cylinder 20 only is recorded. This cylinder is called a file control cylinder and it contains a directory and others. For further information, refer to 2.2.4 File control method.

18 This DISK BASIC controls the area on a disk in the unit called a group. On a RAM disk, one group consists of 16 sectors, and 62 cylinders out of cylinders 1 through 63 excluding Cylinder 20 are divided into 124 groups of 0 through 123 and are used as areas of reading/writing files. However, these areas are not allocated to memories in the initial state. When a RAM disk is used by a disk input/output instruction, if no memory has been allocated to the

group to be read/written, a memory is allocated to the group. The allocated memory is reset when a DSKINI or KILL instruction is executed. For details, refer to 6. Use Method of RAM Disk.

A RAM disk is structured as shown below.

1. N: Unused U: User area F: FAT D: Directory

2.2.4 File control method

All files are controlled in the same way even if they are on different disks, and the following explains the control method.

This DISK BASIC has one reserved cylinder on a disk as a file control cylinder and has FAT (File Allocation Table) and a directory there.

The following shows the configuration of file control cylinder.

1. Configuration of file control cylinder 2. Unused
3. (Note 1) 4. Directory 5. Sector
6. Slot 7 (Note 2)
8. This one bytes shows the use state of Group No. j-1.
9. (See table below.) 10. Meaning of FAT contents
11. Data value 12. Meaning
13. Means this is a part of file and has a succeeding group. The value indicates the succeeding Group No.

14. Means this is the last group of file. A value remained after deducting &HG0 shows the number of sectors actually used.

15. Indicates that when a data file is newly opened, the start of file is secured in the related group.

16. Indicates that this is used by the system and not available to the user.

17. Indicates that this is an unused area.

18. Fine name 19. (Note 3) 20. File name extender

21. t : File type
    0: BASIC program
    1: Data file
    2: Machine language program

    f : Form &H00: Internal expression (binary) form
    &HFF: Character (ASCII) form

    g : Group No. of file head

    s : Number of significant data bytes in the last sector of file

    m : Number of groups per disk

        Mini-floppy disk : 152
        Standard floppy disk: 146
        RAM disk : 124

    d : (m-1) ¥8 + 1

        Mini-floppy disk : 23
        Standard floppy disk: 23
        RAM disk : 20

22. (Note 1) Unused and all bytes are &HFF.

    (Note 2) Unused and all bytes are 0.

(Note 3) If the first one byte is 0, it means the contents have been erased.
If the first one byte is &HFF, it means the end of directory.

There are roughly two ways to access a disk. One is by processing of a file. For this type access, simple instructions only are needed for the start and end of a record, delimiting of individual data items, and the DISK BASIC controls all the rest.
The other type is a direct access to a specific sector by a user program. Use of a DSKO$ or DSKI$ instruction represents this type. However, note that all files must be controlled by individual user as has been explained. Also, if a directory or file is destroyed by DSKO$, a part or whole files of the diskette cannot be accessed. Therefore, extreme care must be paid when using a DSKO$ instruction.

20 2.2.5 Comparison of floppy disks

1. Item No.	2. Comparison item	3. Mini-floppy disk
4. Standard floppy disk	5. RAM disk	6. Cylinder No.
7. Sector No.	8. Group No.	
9. Memory capacity per disk		10. xx bytes xx
11. Number of groups per disk		12. xx groups yy
13. Number of sectors per disk		14 xx sectors xx
15. Memory capacity per group	16. Number of sectors per group	

17. Sector size 18. Memory density of Track 0 on Side 0 (sector size)

19. Double density (256 bytes) 20. Single density (128 bytes)

21. FAT position 22. Cylinder 0, Sector 2 (2nd - 153rd byte)

23. Cylinder 37, Sector 2 (2nd - 147th byte)

24. Cylinder 20, Sector 2 (2nd - 125th byte)

25. Directory position 26. Cylinder 20, Sectors 5 - 23

27. Cylinder 37, Sectors 5 - 23

28. Cylinder 20, Sectors 5 - 20

29. Write protect method 30. Stick the seal.

31. Peel off the seal. 32. Not possible.

33. Note 1) Those in parentheses, ( ), are numerics of user area (including areas used by utilities) in a system diskette.

Note 2) These are limited by the capacity of actually mounted RAM.

Note 3) On a diskette having no write protect notch, the protect notch part must be cut off.

21 3. Utilities

3.1 Starting of utilities

The S1 DISK BASIC provides the following seven utility functions.

(1) Formatting (diskette initialization)

(2) Disk copying

(3) File copying

(4) System copying

(5) Disk dumping (diskette contents display)

(6) System attribute setting

(7) Auto start setting

To use any one of these utilities, first, mount the system diskette to Drive 0 and restart the DISK BASIC by ⌜NEWON 1⌟ instruction. Key in as follows.

RUN "UTILITY"

The following utility menu is displayed on the screen. Key in the number of utility to use. When ⌜0⌟ is input, the system waits for a command of the BASIC.

22 ××××

When using these utilities, the [Y] or [N] key is input against an inquiry. In such a case, the [y] key, [↵] key or [ ] key may be pressed instead of the [Y] key and the [n] key instead of the [N] key.

3.2 Formatting and disk copying

Use these utilities to format a diskette or to make a copy of diskette.

When ⌜1⌟ or ⌜2⌟ is keyed in against the utility menu display, the following submenu is displayed.

××××

(Note) This submenue is displayed when ⌜1⌟ was keyed in.

23 Key in the number of utility to use. If ⌜0⌟ is keyed in, the display returns to the utility menu.

3.2.1 Formatting

A diskette bought in the market must be initialized to the DISK BASIC diskette using this utility before it is used.

When ⌜1⌟ is keyed in on the submenu screen, the following message is output.

××××

Check the diskette to fomat once again, mount it to one of Drives 0 to 3, and input the Drive No. The following message is input.

××××

If, however, Drive No. 0 is input, the following message is output.

××××

Key in ⌜Y⌟ to format. The formatting operation starts.
Key in ⌜N⌟ if the disk is not to be formatted.
When the formatting operation normally completes, the following message is output.

××××

After completing the formatting operation, initialization starts. When the initialization operation terminates, the following message is output.

××××

Then a message of ⌜Continue (Y/N)?⌟ is displayed. If there is another diskette to format, key in ⌜Y⌟ . If not, key in ⌜N⌟ . The display returns to the submenu.

3.2.2 Disk copying

This utility is used to copy the whole contents of one diskette.

(1) 2-drive copying

When ⌜2⌟ is keyed in on the submenue screen, the following message is output.

24

(Note) The source drive is 2 and the target drive is 3 for copying a standard floppy disk.

Set the source diskette to be copied (a diskette that does not contain the DISK BASIC codes nor the utilities may be used) to Drive 0, set a diskette on which a copy is made to Drive 1, and key in ⌜Y⌟. If copying is not wanted, key in ⌜N⌟.

When the copying operation starts, a message notifying that copying is being executed and the Track No. of copying are displayed. When the copying operation normally terminates, the following message is output.

Then, another message of ⌜Continue (Y/N)?⌟ is displayed. Key in ⌜Y⌟ if there is another diskette to copy or ⌜N⌟ if the copying operation is to be terminated.

The display returns to the submenu.

(2) 1-drive copying

To make a copy using one drive only, key in ⌜2⌟ to the submenu screen. The following message is displayed.

Mount the original diskette to Drive 0 and key in ⌜Y⌟. Reading of the original starts and the read track is displayed on the screen by ⌜*⌟. Key in ⌜N⌟ if copying is not wanted.

When the reading normally completes, the message of ⌜Target drive = >0⌟ is displayed. At this time, replace the original diskette on Drive 0 with a diskette on which a copy is to be made, and key in ⌜Y⌟. The read track contents are written in the second diskette. If ⌜N⌟ is keyed in, the copying operation is cancelled.

When the writing normally completes, the system is set back to the state of reading from the original diskette again. Repeat the same operations.

When the copying operation nromally terminate, the following message is output.

××××

This message is followed by another message of ⌜Continue (Y/N)?⌟. Key in ⌜Y⌟ if there is another diskette to copy, or key in ⌜N⌟ if there is none. The display returns to the submenu screen.

3.2.3 Formatting and Disk copying

Use this utility to format a diskette first and to copy the whole diskette continuously.

When ⌜3⌟ is keyed in on the submenu screen, this utility

is started. For details of this utility, refer to 3.2.1 Formatting and 3.2.2 Disk copying.

3.3 System copying

Use this utility to copy a system part (DISK BASIC code) of a system diskette.

When ⌜3⌟ is keyed in to the utility menu, the following submenue is displayed.

26 XXXX

Key in the number of utility to use.

When ⌜0⌟ is keyed in, the display returns to the utility menu.

3.3.1 System copying

Use this utility to copy a system part (DISK BASIC code) of a system diskette.

(1) 2-drive copying

When ⌜1⌟ is keyed in to the submenu, the following message is displayed.

XXXX

At this point, mount a diskette on which a copy is to be made to Drive 1. Also, check that the system diskette is mounted to Drive 0. Key in ⌜Y⌟ . A message informing that the copying operation is being executed is displayed and the copying operation starts. Key in ⌜N⌟ if copying is not wanted.

When the copying operation normally completes, the

following message is output.

27 XYXX

Key in ⌜Y⌟ if there is another diskette to copy, or key in ⌜N⌟ to terminate the copying operation.

(2) 1-drive copying

To make a copy using one drive only, key in ⌜1⌟ to the submenu screen. The following message is output.

XXYX

At this point, check that the original diskette to make a copy is mounted to Drive 0, and key in ⌜Y⌟ . A message is displayed informing that copying is being executed, and reading of the original diskette starts. Key in ⌜N⌟ if copying is not wanted. When the reading normally completes, the following message is displayed.

XXXX

Replace the original diskette mounted to Drive 0 with a diskette on which a copy is to be made and key in ⌜Y⌟ . Writing to the second diskette starts. When the reading normally completes, the following message is displayed.

XXXX

Key in ⌜Y⌟ if there is another diskette to make a copy, or key in ⌜N⌟ to terminate the copying operation.

28 3.3.2 Formatting and System copying

Use this utility to format a diskette first and to copy a system part (DISK BASIC code) of a system diskette continuously.

Key in ⌜2⌟ to the submenu to start this utility. For details of this utility, refer to 3.2.1 Formatting and 3.3.1 System copying.

3.3.3 System copying (with verification)

Use this utility to make a copy while conducting the read after write checking.

Key in ⌜3⌟ to the submenu screen to start this utility.

For details of this utility, refer to 3.3.1 System copying.

3.3.4 Formatting and Disk copying (with verification)

Use this utility to format the diskette on which a copy is to be made and to copy a system diskette while conducting the read after write checking.

Key in ⌜4⌟ to the submenu screen to start this utility.

For details of this utility, refer to 3.2.1 Formatting and 3.3.1 System copying.

3.4 File copying

Use this utility to copy a specified file to another diskette.

Key in ⌜4⌟ to the utility menu display. The following message is output.

XXXX

At this time, key in the file name to copy in the form of ⌜device name: file name. file name extender⌟ , and press the [↵] key. When the device name is omitted, Drive 0 is

set. When the file name extender is omitted, ⌜.DAT⌟ is set. When copying is conducted between mini-floppy disks, if ⌜device name: *.*⌟ or ⌜*.*⌟ is keyed in as the device name, all files are copied. If there is an error in the file name that was keyed in, a message requesting correct key-in is displayed. Input a correct file name.

29 When a file name to make a copy is keyed in, the following message is displayed.

××××

At this point, key in a new file name in the same way as the file name out of which a copy is made and press the ↵ key. If there is a file having the name already, the following message is displayed.

××××

Key in ⌜Y⌟ to make a copy while deleting the existing copy of the name. Key in ⌜N⌟ not to do the copying.

The copying operation starts in the mean while and a message is displayed informing the effect. If the type of drive is different (for example, copying from a mini-floppy disk to a RAM disk), the number of remained sectors is also displayed.

When the file copying completes, the following message is displayed.

××××

Key in ⌜Y⌟ if there is another diskette to make a copy, or key in ⌜N⌟ to terminate the copying operation.

Note that, in order to execute this utility, the DISK BASIC must have been started on a system diskette on which a field buffer size of no smaller than 128 bytes is specified. If the field buffer size is smaller than 128 bytes, first, respecify the size to 128 bytes or greater using the utility of 「System attribute setting」. The field buffer size in the standard state is 256 bytes and re-specification is unnecessary.

3.5 Disk dumping

Use this utility to dump the contents of a specific sector of a diskette or to correct the contents.
Key in 「5」 to the utility menu display. The following message is output in the upper center of the screen.

Also, the following message is displayed at the lower part of the screen, and the system waits for key input.

At this time, specify the sector to dump in the following format. Note that the <drive No.> , <cylinder No.> and <sector No.> are already set to (drive 0, cylinder 20, sector 5) of the first sector of Drive 0 directory.

[<drive No.>][, | [<cylinder No.>][,[<sector no>]] | [,] ¥<group No.> |

][[,] | ± |<number of relative sectors>]

Any specification of the <drive No.> , <cylinder No.> or <sector No.> that is not changed can be omitted. Use a decimal number, hexadecimal number starting with ⌜&H⌟ or octal number starting with ⌜&H⌟ for the specification.

When the <group No.> is specified, it is equivalent to specifying the Cylinder No. and Sector No. of the first sector of the group. Specify with a hexadecimal number without describeing ⌜&H⌟ at the head to the <group No.> .

When the <number of relative sectors> (hereinafter abbreviated as ⌜n⌟) is specified, it means that the specification is made to the sector ahead of the sector specified in the above by n sectors. That is, if +n, it is n sectors ahead and if -n, it is n sectors backward. ⌜n⌟ must be a decimal number. The following shows an example of Drives 0 and 1 are mounted with a mini-floppy disk each, Drives 2 and 3 are mounted with a standard floppy disk each and Drive 4 is mounted with a RAM disk.

1. No. 2. Specification example
3. Specified sector 4. Drive 5. Cylinder
6. Sector

When specification of the sector completes, the display is cleared and the sector contents are dumped to the screen. When the dumpling completes, the following message is

displayed at the lower part of screen, and the system waits for an input.

31 This state of waiting for an input is specially called command waiting state. An example of this state is shown below.

To dump another sector, specify the sector in the format shown earlier.

If there is no need to change the drive, cylinder or sector, the specification for it can be omitted. However, if all three are omitted, it is equivalent to specifying 「+1」, that means the next sector is specified. The current display on the screen is cleared and the contents of specified sector are dumped, and when the dumping ends, the system is set to the command waiting state mentioned above.

Input as follows to revise the display contents.

E[dit] If the first character is 「E」, it is regarded as an EDIT command.

An EDIT command revises the screen display in the same way as the screen editor does. The revised contents are displayed in yellow. The control keys that can be used for the revision are as follows.

|CTRL| + |A| Changes the editing mode (hexadecimal number/character).

[←] [→] [↑] [↓] Use these keys to move the cursor to the position of character to change.

[ ] Use this key to move the cursor by one column in the hexadecimal editing mode.

[HOME CLS] , [SHIFT] + [HOME CLS]

One of these inputs returns the cursor to the home position (upper left of screen). When this key is pressed, the EDIT command terminates and the system is set to the command waiting state.

[CTRL] + [W] This instructs write to the disk. The operation method is the same as the WRITE command, and refer to the WRITE command explanation for further details. However, note that the state returns to the EDIT command when the writing completes.

32 Input as follows to write the revised contents to a disk.

W[RITE] If the first character is 「W」, it is regarded as a WRITE command.

When a WRITE command is input, the following message is displayed and the system is set to the state of waiting for an input.

XXXX

Specify the sector to write into in the same way as the specification of a sector to dump, as explained in the above.

Any parameter that requires no change can be omitted.

When every specification is omitted, it is equivalent to specifying ⌜+0⌟ , and the current sector is written. When specifications are made, the screen displays the specified Cylinder No. and Sector No. to check. The following shows an example of this display.

XXXX

Respond to it as shown below.

[Y] [y] [____] This instructs to write to the sector.

or [↙] Check that writing to it is not protected before keying in this. When the WRITE command terminates, the system returns to the command waiting state.

[N] or [n] This revises the specification of sector to write into.

[Q] or [q] This temporarily stops execution of the WRITE command and sets the system to the command waiting state.

When revising the FAT or directory, sometimes the Cylinder No. and Sector No. must be calculated from the Group No. without changing the contents of current display. In such a case, input as follows.

?<Group No.>

The Cylinder No. and Sector No. of the specified group of the current drive are displayed at the lower part of screen.

After the display, the system returns to the command waiting state.

For the <group No.> , directly write a hexadecimal number immediately after 「?」 without writing 「&H」 .

Input as follows to terminate the disk dumping.

Q[uit] If the first character is 「Q」 , it is regarded as a QUIT command.

When a system diskette is mounted to Drive 0, the display returns to the utility menu. Other than that, the following message is displayed and the system is set to the state of waiting for a BASIC command.

××××

33 Pay attention to the following points when using the system dumping utility.

(1) When the system dumping utility is used to a RAM disk, if a sector in a group that has not been used is dumped, one page of a memory is allocated to the group. This memory cannot be used by other than a RAM disk unless the RAM disk is initialized by a DSKINI instruction. Therefore, care must be paid when specifying a sector of RAM disk.

(2) When specifying a sector to dump or write, even if the specification is made in a format that violates the specification format, sometimes, no error results in when the specified value is correct. In such a

case, respecify the sector in a correct format.

(3) It is not seldom that a cursor moving key is repeatedly pressed during execution of the EDIT command or the [↵] key is repeatedly pressed in the command waiting state. In such a case, since the processing speed of the BASIC does not catch up the repeating speed of the key, the repeating is apt to be given too long , resulting in an overmovement of the cursor. If this happened, press the [PF1] key. The extra pressing of the key remained in the key input buffer is cancelled.

(4) To dump a sector content to a printer, press the [SHIFT] + [COPY] keys while the sector content is displayed on the screen.

(5) Sometimes, disk dumping may not be conducted on the contents of Sectors 0 through 52 of Cylinder 0 of a standard floppy disk that has not been formatted by S1.

3.6 System attribute setting

This utility specifies the largest Drive No. to use, disk type, number of FCB (File Control Blocks), whether or not a RAM disk is used, and whether or not a system call is used.

When 「6」 is keyed in to the utility menu, the following submenu is displayed.

34 ××××

If there is an item to be changed, key in the corresponding No.

The cursor will be positioned to the point to change.

Key in the value to change. When the value to change is keyed in, another message of 「Which (0 ~ 5)?」 is displayed.

If there is another item to change, key in the corresponding No. Key in 「0」 if there is no more item to change. The BASIC codes are rewritten in the system diskette and the display returns to the menu screen. However, if nothing is changed and 「0」 is keyed in, the BASIC codes in the system diskette is not rewritten.

(1) Change of maximum number of drives

Key in the largest Drive No. that can be used. Then key in the disk type for each of the largest number of drives. If an error is made in the specification, an error is made on the utilities of file copying and disk copying.

(2) Change of maximum number of FCB

Key in the number of files that can be opened at a time less one. The number specified here is the number of FCB, but one FCB is needed to open a file and have input/output conducted to a disk. However, since one FCB is reserved in the system, the value to be input at this time should be the number of files to be opened

at a time less one.

(3) Change of field buffer size

The field buffer size means the maximum value of the total of record lengths (fourth parameter of OPEN instruction) of randam files that can be opened at a time. For example, if the field buffer size is set to 512, the record lengths of one random file cannot exceed 512. Also, while it is possible to concurrently open two random files having a record length of 256 each, it is not possible to concurrently open a file having a record length of 256 and another file having a record length of 257 (the file opened later is made an error). Naturally, it is alright to close the first file temporarily and then to open the second file. Also note that this is not related with the number of records on a file. No matter how many records are in a file, if the total length of records is up to the specified field buffer size, the file can be opened.

(4) Setting of RAM disk

Key in 「Y」 to use a RAM disk and key in 「N」 when not using a RAM disk.

When a RAM disk is used, the maximum number of drives plus one is set to the Drive No. If there is an unused RAM area, it can be used instead of a floppy disk.

RAM used in this method is called a RAM disk.

The range that can be used as a RAM disk is 32K bytes to 512K bytes. This RAM disk area is formatted at the time of start, as done on a floppy disk.

A SAVE or LOAD instruction with a file name specification can be handled in the same way as a floppy disk is. Furthermore, since the processing of a RAM disk is faster than a floppy disk, a RAM disk is very useful for data input/output by the BASIC.

(5) Adding of system call

Key in ⌜Y⌟ to add a system call and key in ⌜N⌟ to reset adding of a system call. An explanation of system call will be given on a separate manual.

(Caution)

This utility is effective only when the BASIC codes in the system diskette have been rewritten (therefore, the system diskette must be set to the write enable state beforehand to execute this utility), and it does not affect the DISK BASIC that has been stored in a memory and has been started. After executing this program, the power supply must be turned off and on in order to make the specified values effective.

The system diskette of standard setting (as it is shipped out of the plant), the system attributes are set as follows.

36	Maximum Drive No.	1
	Maximum number of FDC	4
	Field buffer size	256
	RAM disk	none
	System call	Not added.

3.7 Auto start setting

Use this utility to create a system diskette that automatically starts a specified program when the power switch of S1 is turned on.

When ⌜7⌟ is keyed in to the utility menu, the following submenu is displayed.

Key in ⌜1⌟ to set the auto start utility and key in ⌜2⌟ to reset it. The following message is displayed. Mount the system diskette on which the auto start is to be set (or reset) to Drive 0 or Drive 1 and key in the Drive No.

When a Drive No. is keyed in, ⌜OK (Y/N)?⌟ is output. Key in ⌜Y⌟ to set (or reset) auto start. When auto start is set (or reset), the display returns to the submenu. If ⌜N⌟ is keyed in, auto start is not set (or not reset) and the display returns to the submenu.

When ⌜0⌟ is keyed in to the submenu, the display returns to the utilily menu.

(Caution)

Prior to setting auto start, the following program must be created for a file having the name of null string ("") (called unnamed file).

37 (Example)

```
XXXX
```

This is an example of program that automatically starts utility as the power is turned on to S1.

If the above program is not prepared, give the file name of null string ("") to a file that contains a program that should be automatically started.

use ".BAS"

3.8 Error messages of utilities

1. Error message	2. Contents
3.	A CRC (Cyclic Redundancy Check) error or lost data error occured during read from a disk.
4.	An error occurred during read or write of a disk. There is a possibility of the diskette having some damage.
5.	The drive unit to be used is not usable. The drive unit must be made usable by the system attribute setting utility.
6.	Unable to write the diskette mounted to the drive since write protect is given to it.
7.	No diskette is mounted to the drive.
8.	An error occurred during formatting.
9.	The format of original diskette to be copied is not in

the format of this DISK BASIC.

10. An error occurred during disk seek.

11. An error occurred during verify.

12. System diskette not on Drive 0

13. The system diskette is not mounted to Drive 0.

14. Same drive name on Drive n

15. A file having the same name as the new file exists in the diskette mounted to Drive n.

16. Drive n area short

17. The diskette mounted to Drive n has not enough space to copy the specified diskette.

38

1. Error message 2. Contents

3. Not system diskette

4. The diskette mounted to Drive 0 for system copying is not a system diskette.

5. Specification error on [drive][group][track][sector][+n][-n]

6. An error is made on a parameter specified for system dumping.

7. Use this after executing NEWON 1

8. This message is displayed when the unused RAM is shorter than 44 KB at the time of starting utility. (Note)

9. Cannot find file

10. The file having the file name to copy is not in the diskette.

11. (Note) This message may be input when a RAM disk is used even if it has been started by NEWON 1. In such a case. delete unnecessary files from the RAM diskette or executed a DSKINI instruction.

39

4. Language Specifications

This chapter explains instructions and functions that are related with disk access only. Refer to the S1 BASIC manual for details of other instructions and functions.

4.1 Instructions

CHAIN

FUNCTION Calls out programs on a specified file.

FORMAT CHAIN [MERGE]"<file descriptor>" [,[<line number expression>][,ALL][,DELETE [<starting line No.>][,|-|[<ending lane No.>]]]]

STATEMENT EXAMPLE

EXPLANATION

This instruction loads or edits partially and executes the program specified by the <file descriptor>. The difference between this instruction and LOAD, MERGE or RUN statements is that this instruction can relay variables to the program called out.

Specify the Line No. of execution start for the <line number expression>. When this is omitted, the execution

starts at the program head. Different from ordinary Line No., a variable or numeric expression can be used for the <line No. expression>, but, on the other hand, it is not changed even if a RENUM instruction is executed.

When ALL is specified, all variables and arrays are relayed. When it is omitted, the variables and arrays specified by a COMMON statement only are relayed.

When MERGE is specified, a program on the memory and the program called are mixed. At this time, the program to be called must have been saved in the ASCII form. If this is omitted, the called program is loaded as is, and in this case, the program can be in any form. In addition to the ASCII form, a program in the binary form can be used.

The specification of DELETE is used together with MERGE. Specify the range of program to delete using line numbers. These line numbers can be changed by a RENUM instruction.

40 NOTE

(1) A CHAIN statement is processed in the sequence of deletion, merge and execution start. Accordingly, the Line No. to be specified must be a Line No. on the memory (original), and the Line No. specified by the < line No. expression> must be a Line No. of

the program at the time of execution start (after deletion and merge).

(2) This declaration may become invalid depending on how the options are specified, as shown below.

1 format 2. (Note) 3. ○ : Valid

4. X : Invalid

(Note) The part from the program head after the CHAIN statement execution up to the line where a DEFFN statement is must be the same as the program before the CHAIN statement execution. If they are different, the FN function can neigher be used nor redefined.

CLOSE

FUNCTION Closes a file.

FORMAT CLOSE [[#]< file No.> [,[#] < file No.>]

STATEMENT EXAMPLE XXXX

EXPLANATION

This instruction closes the file specified by the < fine No.> and releases the input/output buffer being used at present for the next use. If it is an output file, the CLOSE statement outputs data that remains in the buffer.

Once a CLOSE statement is executed, the specified File No. is no longer associated with the file, and it can be used for a new OPEN statement.

When the specification of <file No.> is omitted, all open files are closed. Also, all files can be closed by executing an END statement, LOAD statement or RUN statement.

41 COMMON

FUNCTION Specifies variables to relay to a program called by a CHAIN statement.

FORMAT COMMON | <variable name> | [, | <variable name> |] ...
| <array name> ( ) | | <array name>( ) |

STATEMENT EXAMPLE

EXPLANATION

This instruction specifies variables and arrays to be relayed to a program called by a CHAIN statement. Variables that are not specified are deleted when the CHAIN statement is executed and released as empty areas.

A COMMON statement is a non-execution statement. It can be written in two or more COMMON statements, but it is not possible to select one COMMON statement or to make one one specific COMMON statement invalid out of a plurality of COMMON statements using an IF statement or others. Also, a COMMON statement must be described before a CHAIN statement.

There should be no variables or arrays having the same name in COMMON statements in one program.

DEVICE

FUNCTION Sets a default device.

FORMAT DEVICE "< device name >"

STATEMENT EXAMPLE ×↙↙<

EXPLANATION

This instruction sets a device name to be used when the specification of device name is omitted on a file descriptor. Immediately after the power is turned on (or execution of a NEW ON instruction), Drive 0 is the default device.

42 DSKINI

FUNCTION Initializes the file control cylinder.

FORMAT DSKINI < drive No. >

STATEMENT EXAMPLE ×↙↙<

EXPLANATION

DSKINI stands for DiSK INItialize, and this instruction initializes the file control cylinder of a diskette (refer to 2.2 Floppy disk).

If the contents of file control cylinder are destroyed, the diskette cannot be used by this system unless DSKINI instruction has been executed to the diskette.

A message of 「Are You sure (Y or N)?」 is displayed on the screen. Check the diskette and drive and input 「Y」. Pressing the [↙] key is unnecessary.

NOTE

A DISKINI instruction to a RAM disk functions slightly differently. Refer to 6. Use Method of RAM Disk.

DSKI$

FUNCTION Reads contents of a disk sector into character strings.

FORMAT DSKI$<drive No.>,<cylinder No.>,<sector No.>, <character variable name 1>[,<character variable name 2>]

STATEMENT EXAMPLE

EXPLANATION

DSKI$ stands for DiSK Input $, and it sets the contents of the specified sector of the specified drive to character strings and assigns 128 bytes of the first half to the <character variable name 1> and 128 bytes of the second half to the <character variable name 2>.

When reading a sector recorded in single density, the specification of <character variable name 2> must be omitted. On the other hand, the specification of <character variable name 2> is essential when reading a sector of double density recording.

Procedures of OPEN or CLOSE are unnecessary.

43 DSKO$

FUNCTION Writes directly to a disk.

FORMAT DSKO$ <drive No.> , <cylinder No.> ,<sector No.> ,
<character expression 1> [, <character expression 2>]

STATEMENT EXAMPLE

EXPLANATION

DSKO$ stands for DiSK Oupput $, and it writes the character code of specified character string to the specified sector of the specified drive, irrevant to the file operation.

Specify 128 bytes of the first half for the < character expression 1> and 128 bytes of the second half for the <character expression 2> . In either case, any part that exceeds 128 bytes is ignored and if it is shorter than 128 bytes, the shortage is filled by characters of character code 0. The contents of < character expression 2> are always written from the 129th byte.

When writing to a sector of single density specification, the specification of < character expression 2> must be omitted. On the other hand, the specification of <character expression 2> is essential when writing to a sector of double density.

Procedures of OPEN and CLOSE are unnecessary.

Be careful that, if a null string ("") is specified for the character expression, abnormal data instead of a

null string is recorded.

FIELD

FUNCTION Defines a field of record.

FORMAT FIELD [#] <file No.> , <field length> AS <character variable name> [, <field length> AS <character variable length>] ...

STATEMENT EXAMPLE

EXPLANATION

This instruction defines the data configuration of a specified random file record. The length of one record of a random file is fixed and the length of each data item in the record is also fixed, and whatever part that exceeds the fixed length is discarded. However, the length of each data area (field) can be specified by an individual user. The data length and corresponding variable name is specified by a FIELD statement.

Each record must not exceed the length specified by an OPEN statement. If the total of <field length> defined
44 by a FIELD statement exceeds the record length specified by an OPEN statement, a "Field Overflow" error results in.

The configuration of the whole records must be defined by one FIELD statement, and the FIELD statement cannot be divided. However, it is possible to define two or

more field configurations to the same record. All these defined field configurations are concurrently effective.

To enter data to a record, a LSET statement, RSET statement or MID$ statement must be used. If a PUT statement is used without executing a LSET or RSET statement, data obtained at the last execution of a PUT statement or GET statememement is written. However, if the data is written from the first PUT statement, the contents are not guaranteed.

If the contents of character variable used for the FIELD statement are changed or assigned by an instruction other than LSET or RSET, the corresponding FIELD statement becomes invalid and the contents of field buffer remain unchanged.

Prior to executing a FIELD statement, an OPEN statement must be executed in the "R" mode to the specified <file No.>

The data of character variable defined by a FIELD statement becomes invalid when the file is closed. If the data is needed after closing the file, assign to other character variable that has not been defined by the FIELD statement and use the assigned character variable.

Note that, if a PUT statement is executed when the total of field lengths is 0, the contents of field buffer at the point are written to a disk.

FILES/LFILES

FUNCTION Outputs a list of file names to the screen or printer.

FORMAT (1) FILES ["<device name>"]

(2) LFILES ["<device name>"]

STATEMENT EXAMPLE XYYX

EXPLANATION

When a disk is specified for the <device name>, the output is in the following format.

1. Fig. 5 LOF and LOC 2. 8 digits 3. 1 digit
4. 3 digits 5. 2 digits 6. <file name>
7. <file name extender> 8. <type> 9. <size>

The meaning of <type> is as follows.

0A: BASIC program file saved in the ASCII form

0B: BASIC program file saved in the binary form

1A: Data file (prepared by a PUT or PRINT# statement)

2B: Machine language program (file prepared by a SAVE statement)

For details, refer to 2.1.3 Types of files.

The <size> expresses a file size by a number of groups.

Format (1) outputs to the screen and Format (2) to a printer. During the output, line feed is conducted as the output width is reached. Therefore, in the case of output to the screen, the display is made into two lines

or three lines depending on the screen width of 40 or characters, but in the case of output to a printer, unless the output width is a multiplication of 20, the format is not controlled correctly. The output width can be changed by a WIDTH statement.

The following shows an output example of setting the output width to 80 characters.

When the specification of <device name> is omitted, a device specified by a DEVICE statement is selected.

GET

FUNCTION Reads records from a random file.

FORMAT GET# <file No.> [, <record No.>]

STATEMENT EXAMPLE

EXPLANATION

This instruction reads the record of specified Record No. out of a random file. The record contents can be referred to by the character variable specified by a FIELD statement.

When the specification of <record No.> is omitted, the LOC function is used as the Record No. The initial value of LOC function is 1, and when the specification of <record No.> is omitted in the first GET statement, it is the same thing as specifying 1 as the Record No.

. 46 Note that prior to executing a GET statement, the file specified by the <fine No.> must have been opened in the random access mode. Also, in order to refer to the read contents, the data configuration must have been defined by a FIELD statement.

INPUT#

FUNCTION Inputs from a sequential file on a disk.

FORMAT INPUT# <file No.> , <variable name> [, <variable name>] ...

STATEMENT EXAMPLE xxxx

EXPLANATION

This instruction reads data from a sequential file from the disk specified by the <file No.> and assigns its contents to a variable.

Prior to executing an INPUT# statement, an OPEN statement must have been executed in the input mode ("I").

A CR (character code 13, &H0D), comma ",", colon ":", or double quotation mark """ is a delimiter of data, and these characters cannot be read as data. Also, both blanks placed before and after a character string are ignored.

KILL

FUNCTION Erases a file.

FORMAT KILL "<file descriptor>

STATEMENT EXAMPLE ××××

EXPLANATION

This instruction deletes a specified file from a disk.

One KILL instruction can only deletes one file. Also, the file to be deleted must be in the open state.

A KILL instruction is effective to disk files of all types (BASIC program file, machine language program file, sequential file and random file).

When the file name extender in the <file descriptor> is omitted, 「.␣␣␣」 (「␣」 is a space) is used.

47 LINE INPUT#

FUNCTION Inputs one line from a sequential file on a disk.

FORMAT LINE INPUT#<file No.>,<character variable name>

STATEMENT EXAMPLE ××××

EXPLANATION

This instruction reads a line from a specified sequential file on a disk into a character variable. Except the following points, this instruction is the same as an INPUT# instruction.

- o One instruction can read only one variable.
- o The data to read must be the character type.
- o All characters up to CR are effective as data.
- o Blanks before and after a character string are both

ignored. CR is not accepted as data.

LIST

FUNCTION Outputs a program list to a specified output file.

FORMAT (1) When no file is specified (output to the screen):

LIST [[<starting line No.>][-|,][<ending line No.>]]]

(2) When a file is specified:

LIST "<file decriptor>" [,[<starting line No.>][-|,][<ending line No.>]]]

STATEMENT EXAMPLE ××××

EXPLANATION

This instruction outputs a part or whole of a program that currently resides on a memory. When the description of <file descriptor> is omitted, the output is made to the screen. Omission of the <starting line No.> is the same thing as specifying the program head, and omission of the <ending line No.> is the same thing as specifying the program end.

A period "." may be used for each of these line numbers, and if it is used, the Line No. pointer memorized by the BASIC as a result of error occurrence is applied.

When a disk is specified for the device name, it is the same as a SAVE instruction in the ASCII form.

When the file name extender in the <file descriptor> is omitted, ⌜.BAS⌟ is applied.

48 LOAD

FUNCTION Reads a program from a disk.

FORMAT LOAD ["< file descriptor >"][,R]

(The abbreviation is LO.)

STATEMENT EXAMPLE XXXX

EXPLANATION

This instruction reads a specified program memory of disk onto a memory. When the specification of < file descriptor> is omitted, Drive 0 (if a DEVICE instruction is executed, the device specified by the instruction) is used and the file name is set to a null string (the same as a space) and the file name extender is set to ⌜.BAS⌟ .

When a LOAD instruction is executed, first, the program on a memory is erased and the variables lose the value, and files are closed if they have not been closed. However, if there is no program to be loaded, the program on the memory is preserved.

When ⌜,R⌟ of the second parameter is specified, the program is executed as soon as the LOAD instruction is executed. At this time, files in the open state are not closed.

When the file name extender in < file descriptor> is

omitted, ⌜.BAS⌟ is applied.

LOADM

FUNCTION Reads a machine language program or data from a disk.

FORMAT LOADM ["< program descriptor> "][,[<offset>][,R]

STATEMENT EXAMPLE

EXPLANATION

This instruction reads a machine language program and others (data recorded by a SAVEM statement) recorded on a file into a memory.

When the <offset> is specified, the reading is made to an address of adding the offset value to the recorded address.

When the option ⌜,R⌟ is specified, the machine language program is executed as soon as the reading completes. The starting address of execution is the address specified by the SAVEM statement.

When the file name extender in <file descriptor> is omitted, ⌜.BIN⌟ is applied.

49 MERGE

FUNCTION Merges a program on a disk with a program on a memory.

FORMAT MERGE "<file descriptor>"[,R]

STATEMENT EXAMPLE XXXX

EXPLANATION

This instruction combines a current program on a memory with a program on a disk and stores the result on a memory.

The program on a disk must be in the ASCII form (refer to the explanations of SAVE and FILES instructions).

The Line No. of program on a disk may be younger than the address of program on a memory or the two can be crossing, but if there are a same Line No. on the two, the line on the memory is erased and the Line No. on the disk is given the priority.

When there is a MERGE instruction in a program, the system returns to the command level after executing the instruction.

When ⌜,R⌟ is described, the program execution starts after executing the MERGE instruction.

When the file name extender in < file descriptor > is omitted, ⌜.BAS⌟ is applied.

NAME

FUNCTION Changes a file name.

FORMAT NAME "< current file descriptor >" AS "< new file descriptor >."

STATEMENT EXAMPLE ××××

EXPLANATION

This instruction changes the name of file on a disk.
If there is a file having the same name as specified by the <new file descriptor> in the sake diskette, a 「File Already Exists」 error results in.
When the file name extender in <file descriptor> is omitted, 「.␣␣␣」 (「␣」is a space) is filled.
When a device name is specified for the <file descriptor>, the same device name must be specified for both of the <current file descriptor> and <new file descriptor>.

50 OPEN

FUNCTION Opens a file on a disk.

FORMAT OPEN <mode>, [#]<file No.>, "<file descriptor>"[, <record length>]

STATEMENT EXAMPLE ××××

EXPLANATION

This instruction conducts initial processing of input/output operation. It defines the File No. of file specified by the <file descriptor>, preserves a buffer used by the file, and sets the file point to the head of opened file.
One of the following three can be used for the <mode>.

1. <mode> 2. Meaning
3. Whether or not the same name file as in disk exists.
4. Yes 5. No

6. Input mode. Used to read a sequential file sequentially from the head.
7. Reads data from the file.
8. An error results in.
9. Output mode. Used to prepare a sequential file anew.
10. Kills the file first and make a new file.
11. Makes a new file.
12. Random access mode. Used to access (read/write) in the record units based on a specified Record No.
13. Access the file.
14. (Note) If a wrong file is specified (for example, specifying a program file), it does not creates an error. If a wrong file was specified, change the file name extender.

Specify a value in a range of 1 ~ 16 for the <file No.>. The specification of <record length> is effective in the random access mode only, and in this case a length that is shorter than the field buffer size specified by the 「system attribute setting」 utility can be specified as the number of characters for one record. If this specification is omitted, 128 is assumed. Also, since the file can be opened by specifying a record length that is different from that at the time of output, care must be paid not to specify a wrong record length.

Instructions such as FIELD, GET, INPUT#, LINE INPUT#, PRINT# and PUT and functions such as EOF, INPUT$ accompanied by a File No., LOC and LOF cannot be used before the relevant file is opened. All other instructions and functions (DSK$, DSKI$, etc.) can be executed without an OPEN statement.

51 The contents of a file in the open state cannot be guaranteed unless it is closed before the diskette is taken out of the drive.

Up to 16 files can be opened at a time, and the number can be changed by applying the 「system attribute setting」 utility. However, in the case of a random file, the total length of records that can be opened at a time cannot exceed the field buffer size.

When the file name extender in <file descriptor> is omitted, 「.DAT」 is applied.

PRINT#

FUNCTION Outputs to a sequential file (format control) on a disk.

FORMAT (1) When an expression gives a numeric:

PRINT# <file No.> [, <expression> | ; / , / ␣ | <expression>]

(2) When an expression gives a character string:

PRINT# <file No.> [, <expression> [; | , | "," | ; | , | ␣]
<expression>] ...]

(␣ is a space.)

(3) When controlling the format:

PRINT# <file No.> , USING <format expression> ;
<expression> [; | , | << expression >] ...

STATEMENT EXAMPLE ××××

EXPLANATION

This instruction outputs data to a specified sequential file.

When USING is used, this instruction can control the format. For explanation of the format expression details, refer to the S1 BASIC manual.

Prior to executing a PRINT# statement, the file must be opened in the output mode ("O"). Also, the data may not be output actually unless the file is closed at the end of processing.

NOTE

The form of output to a disk is about the same as that of output to the screen or printer. (The only difference with a disk is that, since a disk file has no concept of the output width, no line feed control is given even when the output exceeds the output width.)

Therefore, pay attention to the following points when outputting data by PRINT# first and inputting it by INPUT#.

52

(1) When a character string or numeric is to be output immediately after a character string, a comma ⌜,⌟ must be output after the first character string as shown in Format (2). Otherwise, the two character strings are read as one piece of character data when they are read by an INPUT# statement.

(2) When a comma ⌜,⌟ is used as a delimiter of <expression>, the space efficiency of the disk is degraded since the format control is given for each 14 digits.

(3) When a length of one line exceeds 254 digits including the space that is output by the format control of (2), the return and line-feed codes of two bytes are added at the time of line feed, and this makes the line length greater than 255 bytes. Beware that such a line cannot be read by one LINE INPUT# statement (or INPUT# statement).

PUT

FUNCTION Writes a record to a random file.

FORMAT PUT#<file No.> [,<record No.>]

STATEMENT EXAMPLE XXXX

EXPLANATION

This instruction writes data to a specified record of random file. Specify the data configuration by a FIELD statement and set the value using an LSET or RSET statement.

For details, refer to the explanation of FIELD statement.

The <file No.> is the No. of file that was opened in the random access mode.

Specify the location of record in the random file for the <record No.> .

When the specification of <record No.> is omitted, the Record No. next to the record that was accessed by a PUT or GET statement that was executed immediately before is applied, that is, the contents of LOC function are applied.

If a PUT statement is executed in advance to the largest Record No. for recrods sheduled to be used, the necessary disk area is reserved, and other files cannot use the reserved area.

53 RUN

FUNCTION (Loads and) executes a program.

FORMAT RUN [<line No.>] (Abbreviated form is R.)

RUN " <file descriptor> " [,R]

STATEMENT EXAMPLE ××××

EXPLANATION

When the parameter is not specified, the BASIC program is executed in the sequence of lines having a younger Line No. When it is specified, the execution starts from the specified line.

When the <file descriptor> is specified, the specified file on a disk is loaded, and the execution immediately starts. In this case, a program that is existing on the memory is erased.

Prior to executing a program, this instruction clears numeric variable contents to 「0」 and character variable contents to nul strings. It also closes all open files.

However, if the R opetion is specified, the opened files are kept as is and not closed.

When the specification of file name extender in the <file descriptor> is omitted, 「.BAS」 is used.

SAVE

[FUNCTION] Records a file to a disk.

[FORMAT] SAVE ["<file descriptor>" [,|A/P|]]

(The abbreviated form is SA.)

[STATEMENT EXAMPLE] ××××

[EXPLANATION]

This instruction records a program on a memory to the drive specified by the <file descriptor> with a file name.

When 「,A」 is specified as the parameter, the recorded is made in the ASCII form. If it is not specified, the recording is made in the binary form. Specify 「,P」 to prohibit (protect) the saved program from list

display or program editing. For details, refer to the
54 S1 BASIC Manual.

If there is a file having the same name on the specified disk, the file is erased and saved, regardless of it being a program file or data file.

When the specification of file name extender in the <file descriptor> is omitted, ⌜.BAS⌟ is applied.

SAVEM

FUNCTION Records a machine language program or data to a disk.

FORMAT SAVEM "<file descriptor>",<head address>, <last address>,<starting address>

STATEMENT EXAMPLE

EXPLANATION

This instruction saves a machine language program or the like on a memory to the drive specified by the <file descriptor> with a name. Specify the head address, last address and starting address to execute ⌜LOAD "<file descriptor>"„R⌟ of the memory to save.

When the specification of file name extender in the <file descriptor> is omitted, ⌜.BIN⌟ is applied.

VERIFY ON/OFF

FUNCTION Turns on/off write check of a disk.

FORMAT XXXX

EXPLANATION

When a VERIFY ON statement is executed, ⌜read after write check⌟ is conducted at the time of write to a disk and if there is an abnormal point, rewrite is conducted (VERIFY function). When a VERIFY OFF statement is executed, the VERIFY function is reset.

Immmediately after the power switch was turned on or a NEW ON instruction was executed, the system is set to the VERIFY ON state.

This VERIFY function is provided to improve the reliability of data recording. When a VERIFY OFF instruction is executed, the recording speed becomes faster than that of VERIFY ON, but we recommend that the system is used in the VERIFY ON state as much as possible. This VERIFY function has no influence to the data read speed.

55 4.2 Functions

DSKF

FUNCTION Gives a free area of diskette.

FORMAT DSKF (<drive No.>)

STATEMENT EXAMPLE ××××

EXPLANATION

DSKF stands for DiSK Free, and it returns unused area of the diskette contained in the drive specified by the <drive No.> in the units of group.

NOTE

The DSKF function to a RAM disk slightly varies. Refer to 6. Use Method of RAM Disk.

EOF

FUNCTION Determines end of data on a file.

FORMAT EOF (<file No.>)

STATEMENT EXAMPLE ××××

EXPLANATION

EOF stands for End Of File. It is a function that determines whether or not the end of a sequential file is reached and returns the result with a value.
If the data that was read latest is the last data of the file, true (-1) is set. Otherwise Not-true (0) is set. Accordingly, descriptions like 「IF EOF (n) THEN」 or 「WHILE NOT EOF (n)」 are possible.

56 INPUT$

FUNCTION Inputs characters from the keyboard or from an input file.

FORMAT INPUT$ (<number of characters> [,[#] <file No.>])

STATEMENT EXAMPLE ××××

EXPLANATION

This function reads character strings for the specified number of characters from the keyboard or specified

input file. Specify the <number of characters> with an integer in a range of 1 to 255. The input character is not displayed on the screen. Also, all input, other than a key input to temporarily stop the execution, is input as is.

When the specification of <file No.> is omitted, input from the keyboard is taken.

The file specified by the <file No.> must have been opened in the input mode.

LOC

FUNCTION Gives a Record No. to get or put after a random file.

FORMAT LOC (<file No.>)

STATEMENT EXAMPLE XXX

EXPLANATION

LOC stands for LOcation Counter of disk file. It gives a number to the record of next to a record that was given at the end of a raondom file by a GET or PUT statement.

This Record No. is the same number that is taken when the Record No. specification is omitted in a PUT or GET statement.

The initial value of LOC is 1.

If the file specified by the <file No.> has not been opened, an error results in.

The maximum value of LOC is 32767, and after reading the data of Record No. 32767, the LOC value is set to -32768, but if this LOC value is used as a Record No., an error results in.

The LOC value is effective during the period from opening the file of the specified Record No. to closing it. When the file is closed and opened again, the LOC value is set back to 1.

57 LOF

FUNCTION Gives a maximum Record No. of a random file.

FORMAT LOF (<file No.>)

STATEMENT EXAMPLE ××××

EXPLANATION

LOF stands for Length Of disk File. It gives the maximum Record No. of the random file specified by the <file No.> .

The value returned by the LOG function is renewed when a record having a greater Record No. than that is put.

58 5. File Operation

5.1 Program file operation

This clause describes input/output instructions of program files.

Read Program 1. This is a program that registers names, addresses and telephone numbers to a file. We are not

explaining the details of this program at this moment since the explanation is given in 5.2.1 Output of sequential file. We are just using it to explain the operation of program files, and key in Program 1.

Program 1 ×××

First, let's preserve this program on a disk. Key in as follows.

```
SAVE "0:ADDRESS"
```

「ADDRESS」 is the file name and 「0:」 is the device name. The whole of 「"0:ADDRESS"」 is a file descriptor. As explained before, a description of 「SAVE"ADDRESS"」 alone does not clarify whether the file is to be saved to a cassette or to a disk. This is the reason for specifying a device in the file descriptor, and 「0:」 means the disk drive of No. 0. The specification of device name can be omitted when the disk drive of No. 0 is specified only.

Up to eight characters can be used for the file name and all characters other than 「:」 , 「.」 , 「(」 , and 「)」 can be used.

When Program 1 is keyed in, a lamp lights up indicating that the disk is being accessed, the sound of drive head movement is heard, and the program is registered in the disk.

59 Now, Program 1 has been written to a disk. Once a program is saved in a disk, there is no need to keying in the same program again.

Now, we try to call the program out. Erase all programs remianing on the main unit using a NEW instruction. Naturally, even if programs have remained on the main unit memory, they are automatically erased when a LOAD instruction is executed, but in this case, we erase it to confirm that they are called out from the disk. Check that there is no program remained in the main unit using a LIST instruction, and call out the program using a LOAD instruction.

```
LOAD "0:ADDRESS"
```

As explained before, 「"0:ADDRESS"」 is a file descriptor. Have a list output to check that the program has been stored.

The next is how to run this program. Key in:

```
RUN
```

When 「なまえ (name)」 is displsyed, key in a name of somebody. Hiragana, Katakana or alphabetic characters can be used. Then key in his address and telephone number. There will be another display of 「なまえ (name)」 on the line after the next line, input the information of the second person. Continue inputting information for all necessary persons. When everything has been input, key in 「END」 when the 「なまえ(name)」 is displayed. When the last data is written

into the disk, the program execution terminates.

Sample run 1

In addition to applying a LOAD instruction and RUN instruction, a program stored in a disk can be run in two other ways.

(1) LOAD "0:ADDRESS", R

(2) RUN "0:ADDRESS"

Either one of the above executes both of LOAD and RUN instructions.

60 Now, let's use a FILES instruction in order to check that the data just input has been recorded on the disk. When a FILES instruction is executed, the types and names of all files recorded on the disk are known. Key in as follows:

FILES "0:"

"0:" is the file descriptor. However, no file name is needed on this instruction. When this instruction is executed, information on the files contained on the diskette of the specified file is displayed. The file name, file name extender, type and size of all files contained in the diskette are displayed.

Sample run 2

1. Size 2. Type 3. File name extender

A file name extender is a part of a file name in actuality. See Sample run 2.

The 13th file from the head is a utility program that is contined in the system diskette from the beginning, and the next two are newly added files. Both of them are named ⌜ADDRESS⌟ , but the file name extender for the first file is ⌜DAT⌟ and that for the second file is ⌜BAS⌟ . Since a file name extender is a part of a file name, if the file name extender is different, it is a different file.

However, no file name extender was given when SAVE and LOAD instructions were executed and the OPEN statement on Line 1000 of Program 1 has no file name extender. Actually, a file name extender should have been used, but it was omitted. On a SAVE instruction or OPEN statement, the file name extender specification can be omitted. If this specification is omitted on SAVE or LOAD instruction, ⌜BAS⌟ is used and when it is omitted on an OPEN statement, ⌜DAT⌟ is used. In other words, this specification can be omitted if ⌜BAS⌟ is acceptable on a BASIC program, if ⌜BIN⌟ is acceptable on a machine language program and if ⌜DAT⌟ is acceptable on a data file. When specifying it instead of omitting it, other file name extender can be used. This specification cannot be omitted on an instruction that executes file deletion, renaming of file or an instruction that should be executed to any type file. Most of utility programs are given a file name extender called ⌜UTL⌟ ,

and this is because, the file was saved without omitting the file name extender. To specify a file name extender, write a period 「.」 after the end of file name and write up to three characters immediately after the period.

61 Refer to the explanation of FILES instruction in Chapter 4 for other attributes of the instruction.

By the way, the name of 「ADDRESS」 is used on Program 1. If, however, this name is not liked, there is an instruction that can be used to change the name without repeating LOAD and SAVE. Since more address books will be made, probably we should name it as 「ADDRESS1」. Key in as follows:

```
NAME "0:ADDRESS.BAS" AS "0:ADDRESS1. BAS"
```

On a NAME instruction, the specification of file name extender cannot be omitted. The sequence of parameters is the current file descriptor and new name file descriptor.

There is a function called 「DSKF」 which gives the information of remaining capacity in the disk. Key in as follows:

```
PRINT DSKF(0)
```

The numeric displayed shows the number of groups in the empty area of disk. (Note that this is slightly different when a RAM disk is used. Refer to 6. Use Method of RAM disk.)

Use LOADM or SAVEM for input/output of a machine language

program. When allocating a machine language area on a RAM using LOADM, the machine language area must be obtained by a CLEAR statement. For example, to allocate an area from Address &HC000 to Address &HCFFF, execute ⌜CLEAR,&HC000⌟ first and obtain the machine language area. After that, load the program using a LOADM statement. Also, when preparing a machine language program, the area must be obtained by a CLEAR statement in advance.

We mentioned earlier that the specification of device name on file descriptor could be omitted when Drive 0 is used only. However, a DEVICE instruction can be used to change this. When a DEVICE instruction is executed, the device name specified there only can be omitted. Key in as follows:

```
DEVICE "1:"
```

Then, mount a diskette that contains a file to Drive 1, and key in as follows:

```
FILES
```

62 The FILES instruction should have been applied to Drive 1. Now, let's go back to the original state. Key in as follows:

```
DEVICE "01:"
```

Note that, if the specification of file descriptor (device name only) is omitted in a DEVICE instruction, it results in the same thing as giving no instruction.

5.2 SEQUENTIAL ACCESS FILE

5.2.1 Output of Sequential Access File

Program 1 used in 3.2 creates sequential files. This section describes the output instruction of sequential files using Program 1 as an example.

The file is being opened at Line No. 1000. ⌜"O"⌟ of the first parameter is for mode, and O designates the output mode, that is a mode to write into the disk. Comma ⌜,⌟ written the next is the delimit of the parameters. ⌜#1⌟ of the second parameter is for the file number and this designates that the file number of ⌜#1⌟ must be used instead of the file name of ⌜ADDRESS⌟ in succeeding I/O instructions. Therefore, on output instructions to be used from now on, designation of the file name is not needed and the file number is only designation needed. #1 functions to connect the file and buffer at the same time. The buffer numbered as 1 cannot be used for purposes other than outputting the file named as ⌜ADDRESS⌟ until the file is closed later. The last parameter is the file descriptor.

⌜?⌟ is displayed by the INPUT instruction of Line No. 1010, and input from the keyboard is waited for. Character string input is stored in the variable named as ⌜NAME$⌟.

End of the data is determined at the next 1020. If data really ends (if ⌜END⌟ is keyed in), execution goes to Line 1070.

On Line 1030, the address is input, and on Line 1040, the telephone number. These inputs are stored in variables of ADDRESS$ and TEL$ respectively.

On Line 1050, the input name, address and telephone are written into the disk. The first parameter of ⌜#1⌟ is the file number and all others are variable names. Since the

OPEN instruction has already defined that File No. 1 is to be used for output of the file, named as ⌜ADDRESS⌟, in Disk 0, all designation needed here is just the file number. ⌜","⌟ placed between data items is a delimit sign, and without it, total of NAME$, ADDRESS$ and TEL$ is regarded as one data item. ⌜ ;⌟ is also a delimit sign of data items. Difference is ⌜","⌟ is used on the disk and ⌜;⌟ is a delimit sign used on programs. Since the delimit sign on the disk is automatically added at the end of the PRINT# statement, Line 1050 can be divided into the following three lines:

```
1050 PRINT #1,NAMAE$
1051 PRINT #1,ADDRESS$
1052 PRINT #1,TEL$
```

In this case no ⌜","⌟ should be written.

On Line 1060, a space line is given before receiving input for the next person (PRINT instruction), and execution returns to Line 1010 to receiving input again.

From Line 1070 on, execution is for termination processing, and when data end is determined on Line 1020, the execution goes to Line 1070.

Line 1070 closes the file. When terminating file processing, the CLOSE instruction must always be executed. By executing the instruction, the last data or a mark to indicate the file end is written in the disk and the buffer and file connected by the OPEN instruction are separated. Therefore, if the CLOSE instruction is forgotten, the file is not created correctly, and it cannot be read later.

5.2.2 Input of Sequential Access File

Program 2 is a program to read the file prepared by Program 1 and display it on the screen. This section describes input of sequential access files using the Program 2 example.

Program 2

```
2000 OPEN "I",#1,"0:ADDRESS":I=1
2010 IF EOF(1) GOTO 2070
2020 INPUT #1,NAMAE$,ADDRESS$,TEL$
2030 PRINT "なまえ :";NAMAE$
2040 PRINT "じゅうしょ :";ADDRESS$
2050 PRINT "でんわ :";TEL$
2060 PRINT:I=I+1:IF I<7 GOTO 2010
2063 I=1:PRINT "なにかキーをおしてください。 ";
2065 A$=INPUT$(1):PRINT:PRINT:GOTO 2010
2070 CLOSE #1:PRINT"おわりです。"
2080 END
```

First, the file is opened by Line 2000. The parameters are the same as in the case of output, and they are in the order of mode for input or output, file number and file descriptor.

Data end is determined on Line 2010. In the preceding section for output, explanation was given to key in 「END」 to tell data end when all data has been input. It is possible to record a data item having the name of 「END」 on the file to indicate data end, but in the case of input into data file, a convenient function called 「EOF」 is provided. Therefore, let us use it. Contents of the parentheses that follow 「EOF」 indicate the file number. Therefore, what Line 2010 means to jump the execution to Line 2070 when data of No. 1 file terminates.

Omission of this instruction causes an 「Input Past End」 error.

Beware that the file cannot be closed when this error occurs.

This instruction is subjected to a restriction related to the position on the program, that is, the instruction must

be within a loop existing between data read and processing. If it is outside of the loop, it causes an ⌜Input Past End⌟ error even if it is written.

On Line 2020, name, address and telephone number are read from the sequential access file into variables of NAME$, ADDRESS$ and TEL$ respectively. The parameters are written in the order of the file number and variable names to be read into.

What has to be noted at this point is the correspondence between the data and variables to be read into. As explained in 3.1, input is received only in the sequence of output with sequential files. Accordingly, since output in Program 1 was in the order of name, address and telephone number, the same order is followed in reading for input. Therefore, even if Line 2020 is written as follows:

```
2020 INPUT#1,ADDRESS$,NAMAE$,TEL$
```

the sequence of data coming into does not change. This means the name is entered to ADDRESS$ and the address to NAME$.

Furthermore, suppose that the following is added to Line 2025:

```
2025 INPUT#1,NAMAE$
```

What will happen is after reading in one person's name, the next data to be read will start from an address. Accordingly, an address will enter to NAME$, a telephone number to ADDRESS$, and name of the next person to TEL$.

In other words, use of the same variable names in input as used in output does not necessarily mean they will be read into the output variables. Correspondence between data that is read and variables is determined by the sequence of output.

With this explanation, now the reasons should be able to understand that Line 2020 can be written in 3 lines as shown below:

```
2020 INPUT#1,NAMAE$
2022 INPUT#1,ADDRESS$
2024 INPUT#1,TEL$
```

This applies to output also, and writing in one line or writing in two or more lines by splitting the contents result in the same thing.

Sample run 3

```
RUN
なまえ   : ひたち たろう
じゅうしょ : とうきょうと みなとく にしんばし   15-6-24
でんわ   :(03)062-4952

なまえ   : やまぐち ももえ
じゅうしょ : とうきょうと せたがやく いけぶくろ   1-2-3
でんわ   :(03)098-7654

おわりです.
Ready
```

Data that has been input is displayed on the screen by lines from 2030 to 2050.

One space line is placed by Line 2060 and display returns to the input statement of Line 2010.

If there are 7 digits in data of Lines 2060 through 2065, data is collected for each of the 7 digits on the screen, and the next data is displayed upon key input.

When the data terminates, operation jumps to Line 2070 by the instruction on Line 2010. The file is closed by Line 2070. 「#1」is for the file number.

Line 2080 declares end of the program.

The file number in the OPEN statement of Line 2000 needs not be No. 1, but the same file number must be used consistently in the OPEN, INPUT# and CLOSE statements. A number in the range of 1 through 16 can be used for the files.

There are several subjects that were not explained in the section of program file input/output instruction, and here 「MERGE」 is explained. This is an instruction to combine two programs.

Since Program 2 is on the memory right now, let us try to merge a program output from a file created before. One thing that should be remembered is the limitation that programs must have been saved in the ASCII code form if they are to be merged (see EXPLANATION of the FILES instruction). Program 1 has been saved in the internal expression form and it cannot be immediately merged. Therefore, the current program must be saved in the ASCII format first. Key in as shown below:

```
SAVE "0:ｼﾞｭｳｼｮ 2",A
```

The last parameter, 「,A」, designates saving in the ASCII format. No designation of ASCII or internal expression is needed for LOAD.

Then load Program 1.

```
LOAD "0:ｼﾞｭｳｼｮ 1"
```

Now, Program of 「ｼﾞｭｳｼｮ 1」 is on the memory and the program of 「ｼﾞｭｳｼｮ」 2 to merge has been saved in the ASCII format. MERGE can be executed by keying in as follows.

```
MERGE "0:ｼﾞｭｳｼｮ 2"
```

When 「Ready」 is displayed, execute LIST. Program 1 should have been added.

The next operation is to set these two programs to sub-routines, so that they can be registered or displayed at any time desired. First, change Lines 1080 and 2080 as follows to make the two routines to two subroutines.

```
1080 RETURN
2080 RETURN
```

The main routine that controls these two subroutines is in Program 3.

Program 3

```
100 REM じゅうしょろく (シーケンシャル)
110 CLS
120 PRINT TAB(30);"*** じゅうしょろく     ***"
130 LOCATE 10,3
140 PRINT "1 じゅうしょろくの とうろく  "
150 LOCATE 10,6
160 PRINT "2 じゅうしょろくの ひょうじ  "
170 LOCATE 10,9
180 PRINT "3 おわり "
190 LOCATE 10,12
200 INPUT "なにを しますか ";A
210 IF A<1 OR A>3 GOTO 200
220 IF A=3 THEN END
230 ON A GOSUB 1000,2000
240 GOTO 110
```

When this program is run,「なにを しますか?」is displayed on the screen and the system waits for keying in. If「1」is input, the subroutine from Line 1000 is executed and the address catalog file is registered. If「2」is input, file that has been registered is displayed. If「3」is input, the program execution completes and BASIC returns to state of waiting for command.

Sample run 4

```
                    ***  じゅうしょろく        ***

      1 じゅうしょろくの とうろく

      2 じゅうしょろくの ひょうじ

      3 おわり

      なにを しますか ? 2
```

Save this program also. The file name given is「ジュウショ 3」.

Describe the file name of the program that is read from the disk and merged to the current program in the file descriptor of the MERGE instruction.

The program to be read and the current program must not have the same line number. If there is a same line number, the program to be read is given with the priority. For example, if there is a line numbered 100 in the program on the memory and in the program to merge, Line 100 of the current program is erased and contents of Line 100 of the program read from the disk are recorded.

5.2.3 Modification of Sequential Access Files

This section describes addition, correction and erase of data on sequential access files. In the sequential file system, file that is made cannot be modified. Therefore, a new file must be made after correcting the existing file. Program 4 is a program that modifies data of an address catalog file.

Program 4

```
3000 REM***** ファイルを さがす *******
3010 ON ERROR GOTO 3060
3020 NO=1:SEQ=0:PREV=0
3030 OPEN "I",#1,"0:ADDRESS"
3040 GOTO 3110
3050 REM ** "ADDRESS" が ない とき  **
3060 IF ERR <> 63 THEN ON ERROR GOTO 0
3070 RESUME 3080
3080 SEQ=SEQ+1:NO=NO+1:IF NO 2 THEN NO=1
3085 IF SEQ>98 THEN ON ERROR GOTO 0
3090 OPEN "I",#NO,"0:ADRES"+STR$(SEQ)
3100 REM***** つぎの ファイルを さがす *****
3110 ON ERROR GOTO 3180
3120 PREV=NO:SEQ=SEQ+1
3130 NO=NO+1:IF NO>2 THEN NO=1
3140 OPEN "I",#NO,"0:ADRES"+STR$(SEQ)
3150 CLOSE #PREV
3160 GOTO 3120
3170 REM** ファイル が なくなった とき **
3180 IF ERR<>63 THEN ON ERROR GOTO 0
3190 RESUME 3200
3200 REM********* MAIN *********
3210 OPEN "O",#3,"0:ADRES"+STR$(SEQ)
3220 IF EOF(PREV) GOTO 3430
3230 INPUT #PREV,NAMAE$,ADDRESS$,TEL$
3240 PRINT "なまえ :";NAMAE$
3250 PRINT "じゅうしょ :";ADDRESS$
3260 PRINT "でんわ :";TEL$
3270 PRINT "1:そのまま     2:さくじょ      3:ていせい"
3280 INPUT "なにを しますか ";A
3290 IF A<1 OR A>3 GOTO 3270
```

```
3300 ON A GOTO 3410,3220,3320
3310 REM***** ていせい *****
3320 PRINT "1: なまえ   2: じゅうしょ   3: でんわ"
3330 INPUT "なにを ていせいしますか ";B
3340 IF B<1 OR B>3 GOTO 3320
3350 INPUT "どう かえますか ";C$
3360 IF B=1 THEN NAMAE$=C$
3370 IF B=2 THEN ADDRESS$=C$
3380 IF B=3 THEN TEL$=C$
3390 PRINT "これで いいですか ?":GOTO 3240
3400 REM***** レコードを かく *****
3410 PRINT #3,NAMAE$,CHR$(44),ADDRESS$,CHR$(44),TEL$
3420 GOTO 3220
3430 CLOSE #PREV
3440 INPUT "つづけますか(Y/N) ";A$
3450 IF A$="N" GOTO 3520
3460 IF A$<>"Y" GOTO 3440
3470 INPUT "なまえ ";NAMAE$
3480 INPUT "じゅうしょ ";ADDRESS$
3490 INPUT "でんわ ";TEL$
3500 PRINT #3,NAMAE$,CHR$(44),ADDRESS$,CHR$(44),TEL$
3510 PRINT "もっと ";:GOTO 3440
3520 CLOSE #3
3530 END
```

This program is given with varieties of functions to the extent that no other program is needed once a data file is made, but there are two defective points in it. One is that the program has no function to simply display the whole data, and the other is that all members must be displayed even for one addition. Another program that has improved these defects will be shown later. At this moment, just follow the explanation to understand flow of modification steps.

The first operation is to find the latest file. This is performed by Lines 3000 through 3190. Line 3010 will be explained later, and Line 3020 performs initialization of file numbers. ⌜NO⌟ is for the file No. and either 1 or 2 is entered. This program is to make a file of "ADDRES n" and the number that correspond to n is SEQ . When SEQ is 0, no n is attached, and that is the current file "ADDRESS". Line 3030 opens the file.

Since there is a file having the name of ⌜ADDRESS⌟, execution jumps to Line 3110 by the instruction on Line 3040. Explanation of Line 3110 is also held. Line 3120 enters the file No. that opened ⌜ADDRESS⌟ into ⌜PREV⌟ and updates ⌜SEQ⌟ and ⌜NO⌟. Since 1 or 2 only is used for ⌜NO⌟, if the number becomes greater than 2, it is returned to 1 (Line 3130).

Then, Line 3140 opens ⌜ADRES 1⌟ (bacause SEQ is 1). Naturally, there is no such file and ordinarily, it should have caused the ⌜File Not Found⌟ error. The ON ERROR GOTO instruction on Lines 3110 and 3010 prevents the error from occurring. If an error occurs in executing Line 3140, instruction on Line 3110 declared later becomes effective and the operation jumps to Line 3180. Line 3180 checks the error numbers and causes an error should an error of other than ⌜File Not Found⌟. Line 3190 is an instruction that designates a return address of the main routine after error recovery processing, and in this program, operation jumps to Line 3210. Since the current SEQ is 1, Line 3210 opens ⌜ADRES 1⌟ in the output mode.

Next question is what to do for modification next time after the current modification. Since ⌜ADDRESS⌟ and ⌜ADRES 1⌟ exist now, a new file of ⌜ADRES 2⌟ must be opened. In this case, no error occurs in the preceding Line 3140 and the operation goes to Line 3150. Since File No. 1 that

opened ⌜ADDRESS⌟ is in PREV , Line 3150 closes PREV . Then the operation returns to Line 3120, saves the current file number and then updates the file number and SEQ trying to open ⌜ADRES 2⌟. If the file has not been prepared, it causes an error and the operation jumps to Line 3180, entering the main routine, and opens ⌜ADRES 2⌟ as a new file.

All in all, on operations of Lines 3110 through 3190, SEQ is updated until the ⌜File Not Found⌟ error occurs, thus obtaining the SEQ value to open a new file.

Explanation thus has been given assumes presence of a file called ⌜ADDRESS⌟. What will happen if there is no such file. Processing for such case is covered by Line 3010 and Lines 3050 through 3090. If there is no file named as ⌜ADDRESS⌟, the ⌜File Not Found⌟ error occurs on Line 3030, and operation jumps from Line 3010 to Line 3060. Line 3060 checks the error code, and Line 3080 updates SEQ and NO. No PREV is needed here. Entered in PREV is the file number that has opened the existing file. Line 3090 opens the file of the next SEQ. If such file does not exist, operation returns to Line 3060. In other words, through operation of Lines 3060 to 3090, a real file having the smallest SEQ is searched. When such file is discovered, operation changes to search a file having the last SEQ from Line 3110. If neither of ⌜ADDRESS⌟ nor ⌜ADRES n⌟ exists, operation enters an indefinite loop, which is checked by Line 3085.

The above is the program flow from Lines 3000 through 3190. What it does are to open a file having the greatest SEQ, enter the file number to PREV and obtain SEQ of a file to be created anew. It is very complicated but read the program and understand the contents well.

As to the main routine starting from Line 3210, a file having new SEQ is opened in the output mode by Line 3210, as explained before. Data is input from the last file by Line 3230. Since ⌜NO⌟ is the file number of the file that could not be opened, the file number that reads in data must be PREV.

Through operation of Lines 3240 to 3290, data is displayed and input for instruction of what to do is received. The operation jumps to a processing routine as input by Line 3300. If ⌜1⌟ is input, operation goes to Line 3410 and prints the data as is. If ⌜2⌟ is input, since 2 means erase, no data is output and the operation jumps to read of the next data from Line 3220. If ⌜3⌟ is input, the operation goes to the correction routine from the next Line 3320.

The correction routine is covered by Lines 3320 through 3390. First, what and how the correction should be made is inquired by Lines 3320 through 3350, the correction is made by Lines 3360 through 3380, and the operation returns to Line 3240 to display the results, and enters the state of waiting for input. If correction is made as expected, designate ⌜1⌟ as is. If there is an input error or correction on other item is needed, designate ⌜3⌟ again.

Data corrected by Line 3410, or data as input, is output, the operation returns to Line 3220 by Line 3420 and waits for data input. For this line number and the second line of Line 3300, Line 3320 must be designated. If 3230 is designated, reading out the data to be read causes an error.

When there is no more data to read in the old file, the operation goes to Line 3430 by Line 3220, which closes the old file and an inquiry is made whether or not to add.

If no addition is needed, the operation jumps to Line 3520 and closes the file, terminating the correction routine. Through operation of Lines 3470 to 3490, data to be added is received, and Line 3500 outputs the added data. When adding completes, the operation jumps to Line 3440 and inquires if there is further addition.

The key part of this program is the operation of opening the latest existing file, performed at first, and opening of a file to be created anew. Files where the number assigned to the file name is incremented at each file update are called generation files, and this type of file management is called generation management. It is a method frequently used in sequential file management of large computers.

Program 4 is a program which has the function to display the whole data but not to display other data items when modification is only addition of data.

Program 4

```
10 REM************************
20 REM** じゅうしょろく(シーケンシャル)  **
30 REM************************
40 REM
100 REM***** メニュー *****
110 CLS
120 PRINT TAB(25);"*** じゅうしょろく    ***"
130 LOCATE 10,3
140 PRINT "1:ひょうじのみ"
150 LOCATE 10,6
160 PRINT "2:ついかのみ"
170 LOCATE 10,9
180 PRINT "3:ていせい,さくじょ,ついか"
190 LOCATE 10,12
```

```
200 PRINT "4:おわり"
210 LOCATE 10,15
220 INPUT "なにをしますか ";A
230 IF A<1 OR A>4 GOTO 220
240 IF A=4 THEN END
250 ON A GOSUB 2000,1000,3000
260 GOTO 110
1000 REM***** ついかのみ *****
1010 PRINT "しょうしょうおまちください。"
1020 GOSUB 4000
1030 OPEN "O",#3,"0:ADRES"+STR$(SEQ)
1040 IF EOF(PREV) GOTO 1080
1050 INPUT #PREV,NAMAE$,ADDRESS$,TEL$
1060 PRINT #3,NAMAE$,CHR$(44),ADDRESS$,CHR$(44),TEL$
1070 GOTO 1040
1080 CLOSE #PREV
1090 GOSUB 4600
1100 IF A$="Y" GOTO 1090
1110 CLOSE #3
1120 RETURN
2000 REM***** ひょうじのみ *****
2010 GOSUB 4000
2015 I=1
2020 IF EOF(PREV) GOTO 2050
2030 GOSUB 4500
2032 I=I+1:IF I<7 GOTO 2020
2035 PRINT "なにかキーをおしてください。";
2040 A$=INPUT$(1):PRINT:PRINT:GOTO 2015
2050 CLOSE #PREV
2053 PRINT "おわりです。        なにかキーをおしてください。"
2060 A$=INPUT$(1):RETURN
3000 REM***** てすと, さくじょ, ついか *****
3010 GOSUB 4000
3020 OPEN "O",#3,"0:ADRES"+STR$(SEQ)
```

```
3030 IF EOF(PREV) GOTO 3400
3040 GOSUB 4500
3050 PRINT "1:そのまま     2:さくじょ     3:ていせい";
3060 PRINT "      4:このレコードのまえについか"
3070 INPUT "なにをしますか";A
3080 IF A<1 OR A>4 GOTO 3050
3090 ON A GOTO 3300,3030,3110,3210
3100 REM***** ていせい *****
3110 PRINT "1:なまえ    2:じゅうしょ    3:でんわ"
3120 INPUT "なにを ていせいしますか";B
3130 IF B<1 OR B>3 GOTO 3110
3140 INPUT "どう かえますか";C$
3150 IF B=1 THEN NAMAE$=C$
3160 IF B=2 THEN ADDRESS$=C$
3170 IF B=3 THEN TEL$=C$
3180 PRINT "これで いいですか ?"
3190 GOSUB 4520:GOTO 3050
3200 REM***** まえに ついか *****
3210 GOSUB 4600
3220 IF A$="Y" GOTO 3210
3230 GOSUB 4520:GOTO 3050
3290 REM***** そのまま かく *****
3300 PRINT #3,NAMA#$,CHR$(44),ADDRESS$,CHR$(44),TEL$
3310 GOTO 3030
3400 REM** ていせい さくじょ ついか おわり    **
3410 CLOSE #PREV
3420 GOSUB 4730
3430 IF A$="Y" THEN GOSUB 4600:GOTO 3430
3440 CLOSE #3:RETURN
4000 REM***** さいしん ファイルの OPEN *****
4010 REM
4020 REM*** ファイルを さがす ***
4030 ON ERROR GOTO 4080
4040 PREV=0:SEQ=0:NO=1
```

```
4050 OPEN "I",#1,"0:ADDRESS"
4060 GOTO 4140
4070 REM ** "ADDRESS"が ないとき **
4080 IF ERR <> 63 THEN ON ERROR GOTO 0
4090 RESUME 4100
4100 SEQ=SEQ+1:NO=NO+1:IF NO>2 THEN NO=1
4110 IF SEQ>98 THEN ON ERROR GOTO 0
4120 OPEN "I",#NO,"0:ADRES"+STR$(SEQ)
4130 REM***** さいごの ファイルを さがす *****
4140 ON ERROR GOTO 4250
4150 PREV=NO:SEQ=SEQ+L
4160 NO=NO+1:IF NO>2 THEN NO=1
4170 IF SEQ<100 GOTO 4210
4180 CLOSE
4190 PRINT "SEQが OVERFLOW しました。"
4200 ERROR 21
4210 OPEN "I",#NO,"0:ADRES"+STR$(SEQ)
4220 CLOSE #PREV
4230 GOTO 4150
4240 REM** ファイルが なくなったとき **
4250 IF ERR<>63 THEN ON ERROR GOTO 0
4260 RESUME 4270
4270 ON ERROR GOTO 0
4280 RETURN
4500 REM***** 1レコード よんで ひょうじ する *****
4510 INPUT #PREV,NAMAE$,ADDRESS$,TEL$
4520 PRINT "なまえ :";NAMAE$
4530 PRINT "じゅうしょ :";ADDRESS$
4540 PRINT "でんわ :";TEL$
4550 PRINT
4560 RETURN
4600 REM****1レコード ついか ****
4610 PRINT "ついかデータを にゅうりょく して ください。"
4620 INPUT "なまえ ";N$
```

```
4630 INPUT "じゅうしょ ";N$
4640 INPUT "でんわ  ";AD$
4650 PRINT "なまえ :";N$
4660 PRINT "じゅうしょ :";AD$
4670 PRINT "でんわ :";T$
4680 INPUT "これでいいですか(Y/N) ";A$
4690 IF A$="N" GOTO 4610
4700 IF A$<>"Y" GOTO 4680
4710 PRINT #3,N$,CHR$(44),AD$,CHR$(44),TS
4720 PRINT "もっと";
4730 INPUT "つづけますか(Y/N) ";A$
4740 IF A$<>"Y" AND A$<>"N" GOTO 4720
4750 RETURN
```

Explanation of Program 4 is omitted since it is a combination of the preceding programs. Save this program as 「ジュウショ 4」.

One caution is given for using this program, which is n , (「n」 of 「ADRES n」), of existing files must be continuous.

For example, if there are files named 「ADRES 2」 , 「ADRES 3」, and 「ADRES 5」, but no file of 「ADRES 4」 exists as it has been KILLed, the program would recognize the file of 「ADRES 3」 as the latest file. If a file is deleted, all the preceding files must also be deleted.

One more explanation regarding instructions that relate to file handling. Execute the FILES instruction. Files of 「ジュウショ 1」, 「ジュウショ 2」, 「ジュウショ 3」, and 「ジュウショ 4」 will be displayed as program files (files of Kind 0). Files of 「ジュウショ 2」 and 「ジュウショ 3」 are no longer needed as long as there are files of 「ジュウショ 1」 and 「ジュウショ 4」. Therefore, the two program can be deleted from the registration, and for the purpose the KILL instruction is used as follows:

```
KILL "0:ｼﾞｭｳｼｮ 2"
KILL "0:ｼﾞｭｳｼｮ 3"
```

Parameter needed for the KILL instruction is for the file descriptor only.

5.2.4 Summary of Sequential Access File

The following are advantageous points of sequential access files:

(1) Easy programming

(2) Record definition is left with users completely.

On the other hand, they have the following disadvantages:

(1) It requires a long time to find a specific record.

(2) The OPEN and CLOSE instructions must be repeated each time mode for write and read is changed.

(3) Data that has been registered cannot be erased, corrected nor added.

(4) Because of the above, the processing is slow when compared with random access files.

On ordinary file processing, inconvenience caused by these inferior points are more noticeable and users often think that they are of no use. To find out a specific record, files must be read from the start. After finding one record, the file must be CLOSED and OPENed again to find another record. There is a method of reading all data in arrays, but this leaves problems such as what should be the adequate element number of the array, memory capacity, etc., and the method is not realistic.

To improve the third defective point, a file that was made before must be read and a new file must be made while giving due corrections. This procedure, however, may be an

essential point since if an error is made in directly correcting an old file, correct data will have been lost.

It is possible to leave files before correction in the case of random access files, but this is not the way to fully utilize the advantage of random access files. In either case, if leaving files before correction is essential, sequential access files provide easiness in programming.

In addition, the record length is fixed in random access files and data items longer than the length must be cut. Very little can be done if the maximum number of items cannot be set. On this point, sequential access files are far superior. Though inconvenient, sequential access files permit realization of all functions.

Nevertheless, for processing simple items like address catalogs, random access files, of which explanation will be given in the next section, seem to be better fitted.

There are other instructions related to handling sequential access files, in addition to those explained so far, but they are scarecely used and explanation is omitted.

5.3 RANDOM ACCESS FILE

5.3.1 Output of Random Access File

This section describes instructions that output random access files.

Program 1 for sequential access files can be rewritten for random access files as shown by Program 5 below.

Program 5

```
1000 OPEN "R",#1,"0:じゅうしょろく "
1010 FIELD #1,15 AS N$,40 AS A$,13 AS T$,2 AS Y$
1020 R=1
1030 INPUT "なまえ  ";NAMAE$
1040 IF NAMAE$="END" GOTO 1150
1050 INPUT "じゅうしょ  ";ADDRESS$
1060 INPUT "でんわ  ";TEL$
1070 INPUT "ねんれい  ";YEAR
1080 LSET N$=NAMAE$
1090 LSET A$=ADDRESS$
1100 LSET T$=TEL$
1110 RSET Y$=MKI$(YEAR)
1120 PUT #1,R
1130 R=R+1
1140 GOTO 1030
1150 CLOSE #1
1160 END
```

How does it compare? They look very different from Program 1, but the functions are completely same. One additional item given in this program is the age, and all others remain same. There are a number of instructions that are different from those of Program 1. Let us go over it line by line.

Line 1000 is the OPEN statement, which is needed in any program. This OPEN statement is the same as that of sequential access files, but the input/output mode is ⌜"R"⌟. In random access files, all of read, addition and correction can be operated with one OPEN-CLOSE.

Since the file name of ⌜ADDRESS⌟ was used in 3.3, a different file name is needed here.

The FIELD instruction on Line 1010 did not exist in sequential access files. In random files, the record configuration must be clearly defined. Random files provides a new function that was not available with sequential files and that is read and write in units of record, and a number called Record No. is used as the key.

Location of specific records in the disk must be calculated from the record number, and it can be easily understood that size of each record must be fixed to allow the calculation. However, individual data items are not necessarily in the same length. Therefore, how to allocate the excess parts must be specified in advance. The FIELD instruction performs the specification, that is, it determines the file buffer partition.

The first parameter is for file number and designations of data area size and character variables follow to it. ⌜15 AS N$⌟ designates that a space for 15 characters is assigned to the N$ character variable. Rest of the line designates assignment of 40-character space to A$, 13-character space to T$ and 2-character space to Y$. To summarize this, one record is structured as shown in Fig. 3.

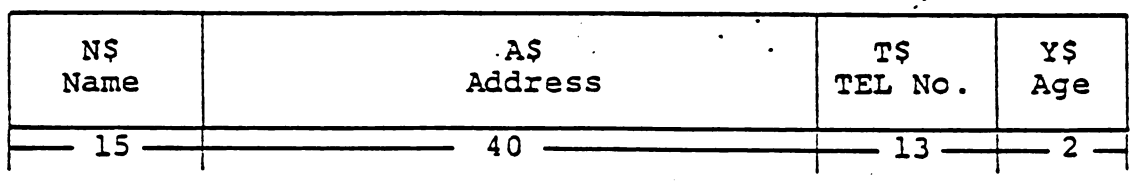


Fig. 3 Record Structure

Variables used in the FIELD statement must be character variables. Character variables must be designated for even numerics, which will be explained later. Also be careful that variables used in the FIELD statement must not be used in instruction statements that alter the or assign the contents (except LSET asd RSET instructions).

Line 1020 initializes the record number.

On Lines 1030 through 1070, data input is received. A strange instruction appears on Line 1080. The LSET instruction is akin to the LET statement. Data is entered in a file buffer by the LSET statement. LSET or RSET is used depending how the data is to be entered. LSET is for left justification and RSET for right justification. Suggested use of these instructions if LSET for name and character and RSET for money amount and age.

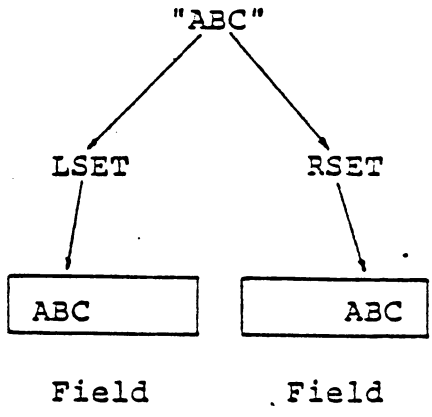


Fig. 4 LSET and RSET

By the way, you may think that receiving input with the INPUT statement and assigning the input to the buffer with LSET or RSET are double actions, but this is inevitable. Variable names defined in the FIELD statement cannot be used in the INPUT statement and data cannot be entered in the buffer by defining with the FIELD statement only, and without LSET or RSET statement.

Lines 1090 and 1100 are both LSET statements.
Line 1110 is RSET statement. MKI$ is a function and it converts numeric data to character type data.

Data registered in random files must be character type data always. Therefore, a function that converts numeric data to character data is needed.

There is another function, STR$, that converts numerics to characters for PRINT statement. The difference between STR$ and MKI$ is that STR$ converts numerics to ASCII codes of decimal notation where as MKI$ converts data type of internal expression only without changing the contents. For example, 20 in decimal notation is 14 in hexadecimal notation and if it is converted by STR$, it becomes character strings of 32, 30 (hexadecimal in both cases) of ASCII codes. If it is converted by MKI$, the contents of data, 14, remains as is and what is changed in only the index that indicates the data type.

Also note that there are differences in the functions when numerics are converted to characters for random files, depending on if it is integer, single precision real number, or double precision real number. MKI$ is a function used for integers and MKS$ is used for single precision real numbers and MKD$ for double precision real numbers. By these functions, an integer is converted to character data of 2 bytes, single precision real number to that of 4 bytes and double precision real number to that of 8 bytes.

Character strings converted by these functions can be converted back to the original numerics by CVI, CVS and CVD functions respectively, which will be explained in the next section.

PUT on Line 1120 is an output instruction. 「#1」 is for file number, 「R」 for record number and where the data is written is determined by these.

Line 1130 updates the record number.

Line 1140 sets the operation to data input again.

Since correspondence between variables and data is defined in the FIELD statement in random files, delimit signs as used sequential files are not needed.

Line 1150 closes the file. Function of the CLOSE instruction is same to both sequential and random files.

Program 5 can be used in the same way as Program 1, but there is no need of preparing the same file as was prepared in Section 3.3. Program 6 is a program that converts the sequential file created in Section 3.3 to a random file. Beware that use of Program 6 after using Program 5 will erase the file created by Program 5 completely since the file to be created bears the same file name. Or, change the file name of Program 5 appearing on Line 1000. Do not change the file name of Program 6, as otherwise all the following programs in this manual must be changed. If a 「ひゃうはぶく」 file has already been made, change the file name using the NAME instruction.

Program 6

```
10 OPEN "I",#1,"0:ADDRESS"
20 R=1:OPEN "R",#2,"0:じゅうしょろく"
30 FIELD #2,15 AS N$,40 AS A$,13 AS T$,2 AS Y$
40 IF EOF(1) GOTO 140
50 INPUT #1,NAMAE$,ADDRESS$,TEL$
60 PRINT NAMAE$;"  ";ADDRESS$
65 PRINT TAB(10)
70 INPUT "ねんれい ";YEAR
80 LSET N$=NAMAE$
90 LSET A$=ADDRESS$
100 LSET T$=TEL$
110 RSET Y$=MKI$(YEAR)
120 PUT #2,R
130 R=R+1:PRINT:GOTO 40
140 CLOSE #1,#2
150 END
```

Lines 10 and 20 open the files, and the sequential file is numbered as 1 and the random file as 2. Therefore, the file number on Line 30 must be 2. Line 20 initializes the record number also.

Line 40 determines data end and Line 50 reads data. Since the sequential file does not contain ages, after displaying names and addresses by Line 50, key in is received by Line 70.

Sample run 5

```
RUN
ひだら たろう   とうきょうと みなとく にしんばし 15-6-24
                ねんれい ? 27
やまぐち あやえ   とうきょうと せたがやく いけぶくろ 1-2-3
                ねんれい ? 23
はぎわら ともこ   とうきょうと としまく ★★★ 9-8-7
                ねんれい ? 32
```

Lines 80 through 110 set the data to the buffer and Line 120 writes the data.

Line 130 updates the record number and operation returns to data input again.

When there is no data to input, the operation terminates closing the file with Line 140. When the file number in the CLOSE statement is omitted, all files in open state are closed.

5.3.2 Input of Random Access File

Program 7 is an input program of random access files.

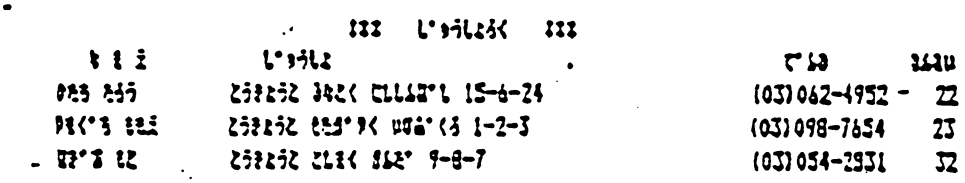
Program 7

```
2000 R=0:CLS
2010 OPEN "R",#1,"0:じゅうしょろく"
2020 FIELD #1,15 AS N$,40 AS A$,13 AS T$,2 AS Y$
2030 PRINT TAB(25);"***  じゅうしょろく  ***"
2040 PRINT "  なまえ  ";TAB(18);"じゅうしょ";
2045 PRINT TAB(63);"でんわ ";TAB(74);"ねんれい"
2050 FOR I=1 TO 22
2060 IF R>=LOF(1) THEN I=24:GOTO 2090
2070 R=R+1:GET #1,R
2080 PRINT N$;TAB(15);A$;TAB(60);T$;TAB(75);CVI(Y$)
2090 NEXT I
2100 IF I>23 GOTO 2120
2110 PRINT "なにかキーをおしてください";:B$=INPUT$(1)
2115 PRINT:GOTO 2030
2120 CLOSE #1
2130 END
```

Line 2000 is for initialization of the record number. Line 2010 opens the file. Line 2020 sets the FIELD statement, which is needed for programs to only read files.

Lines 2030 and 2040 write the heading and Lines 2050 through 2090 read and display the data, and processing here is rather complicated.

Sample run 6



25 lines can be displayed on the screen, but since 2 lines are used for the heading and 1 line at the end for waiting for input, when data for 22 lines is displayed, the display stops and the system waits for input. Only one line is used for one record.

There are 2 purposes for designating 24 for I on Line 2060 and that is, to terminate the FOR-NEXT loop when the data ends and to know if the data ended when operation comes out of the FOR-NEXT loop. If data has not ended by Line 2100, the FOR-NEXT loop is executed in the state of I being 22, and then I becomes 23 by NEXT I on Line 2090. If data has ended, I is changed to 24 by Line 2060 and the operation jumps to Line 2090 and I becomes 25 by ⌜NEXT I⌟. Therefore, if I is greater than 23, it means that there is no more data.

Line 2080 displays data, but as you remember, in the preceding section it was said that variable names defined in the FIELD statement cannot be used in the INPUT statement.

This is because statement like PUT and GET do not function properly with variables defined in the FIELD statement unless LSET or RSET is used to define or alter the contents but in the case of the PRINT instruction, since it is not an instruction that changes the contents, use of these variables does not bring any inconvenience.

The function of CVI appears here as predicted in the preceding section. At the time MKI$ was used and the following is review of the function.

(1) Data to be stored in random file buffers by LSET or RSET must be in character type.

(2) MKI$ is used to convert integer data to character type, MKS$ for single precision real number data and MKD$ for double precision real number data.

The following summarizes condisions for CVI:

(1) If data read into a random file buffer by the GET statement was originally a numeric, CVI, CVS or CVD is used to convert it back to the original numeric data.

(2) If the original data was an integer, use CVI to convert the character data to the original numeric. In the same way, use CVS in the case of single precision real number data and CVD in the case of double precision real number data.

This explanation can be rephrased as follows.

(2)' Data converted by MKI$, data converted by MKS$ and data converted by MKD$ can be converted back to the original type by using CVI, CVS and CVD respectively. I stands for Integer, S for Single and D for Double.

If data has not ended, the system waits for input as stated in Line 2110. If the [RETURN] key is pressed, the operation

returns to displaying the heading as programmed on Line 2030.

If data has ended, the file is closed and the operation terminates.

A point to be noted in the FIELD statement at the time of input is that the number of characters of each data item must concur with that of output program. The variable names need not be the same. Other than these points, the input program is simpler than the output program.

The next explanation on read/write in units of record, which is one of the characteristics of random access files.

5.3.3 Modification of Random Access File

This section describes modification of random access files which involves addition, correction and deletion of data.

With sequential access files, addition, correction and deletion of data cannot be made directly to the files, and what has to be done is to read the file before modification, give necessary modification and to create a new file after the modification. On random access files, it is possible to give necessary modification after reading a specific record and put it back to the same file. To instruct the computer which record to be read and where to store the modified record, record numbers are used.

To add data, designate the next number of the last record of the current file as the record number. To correct or delete data, designate the record number to which the processing is needed, and designate the same record number as before.

Doing all these at a time complicates the program. Therefore, the first example, Program 8, is for adding data.

Program 8

```
3000 OPEN "R",#1,"じゅうしょろく"
3010 FIELD #1,15 AS N$,40 AS A$,13 AS T$,2 AS Y$
3020 R=LOF(1)
3030 INPUT "なまえ  ";NANAE$
3040 IF NAMAE$="END" GOTO 3120
3050 INPUT "じゅうしょ ";ADDRESS$
3060 INPUT "でんわ  ";TEL$
3070 INPUT "ねんれい ";YEAR
3080 LSET N$=NAMAE$:LSET A$=ADDRESS$
3090 LSET T$=TEL$:RSET Y$=MKI$(YEAR)
3100 R=R+1:PUT #1,R
3110 PRINT:GOTO 3030
3120 CLOSE #1:END
```

Other than Line 3020, Program 8 is completely same as Program 5, an output program, that is, initialization of the record number is the only different point.

The function of LOF, which by the way is closer to a system variable rather than being a function, is set by the OPEN instruction. Give the largest number of the record numbers in the designated file. This does not necessarily mean the number of record that was written latest, as records having larger number can be written in before younger number records are written. When this feature is utilized, record management like assigning record numbers up to 50 to names starting with A, from 51 to 100 to names starting with B, and from 101 up to 150 to names starting with C, etc. However, if the record system is made in such way, the program to be prepared for adding data cannot be so simple as Program 7.

There is another function that relates to record numbers, and that is LOC. The name is similar to LOF, so is the

meaning, and care must be paid not to mix the two.

For the function of LOC, give the next number of the record that was accessed latest. In the GET and PUT statements, the record number can be omitted, and if it is omitted, the result is the same as designating the next number of the record that was accessed latest. In other words, for sequential access like Programs 5, 6 and 7, there is no need of assigning a record number. (However, be sure to understand that in read/write operation which is explained from now on, access is not necessarily in the sequence of record numbers, and record numbers must be designated.) LOC is a record number that is used when the record number is omitted.

Once again, we call your attention not to mix LOF and LOC as they are spelled very similar.

The next explanation is on a program for correction and deletion in units of record. See Program 9.

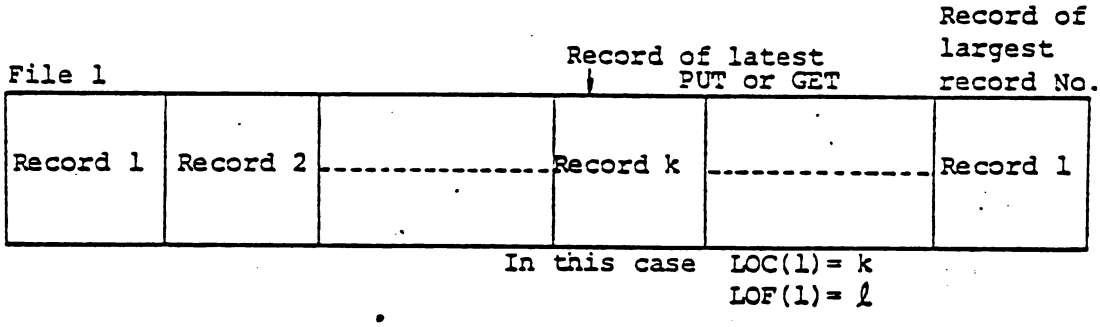


Fig. 5 LOF and LOC

Program 9

```
3500 OPEN "R",#1,"じゅうしょろく"
3510 FIELD #1,15 AS N$,40 AS A$,13 AS T$,2 AS Y$
3520 INPUT "レコード NO.";R:IF R<=LOF(1) GOTO 3530
3525 PRINT "レコード NOは";LOF(1);"までです ":GOTO 3520
```

```
3530 GET #1,R
3540 PRINT "なまえ";N$;TAB(25);"じゅうしょ:";A$;TAB(20);
3550 PRINT "でんわ:";T$;TAB(43);"ねんれい:";CVI(Y$)
3560 INPUT "1:ていせい  2:さくじょ  3:なにもしない";B
3570 IF B<1 OR B>3 GOTO 3560
3580 IF B=3 GOTO 3600
3590 ON B GOSUB 3700,3900
3600 PUT #1,R
3610 PRINT " NEXT (Y/N)?";
3620 B$=INKEY$:IF B$="Y" THEN PRINT:GOTO 3250
3630 IF B$<>"N" GOTO 3620
3640 CLOSE #1:END
3700 REM **** ていせい ****
3710 PRINT TAB(5);"1:なまえ  2:じゅうしょ  3:でんわ";
3720 PRINT TAB(39);"4:ねんれい"
3730 B$=INKEY$:IF B$="" GOTO 3730
3735 IFB$<>"1"ANDB$<>"2"ANDB$<>"3"ANDB$<>"4"GOTO3730
3740 PRINT:PRINT "どう かえますか";
3750 IF B$="4" THEN INPUT C ELSE INPUT C$
3760 IF B$="1" THEN LSET N$=C$
3770 IF B$="2" THEN LSET A$=C$
3780 IF B$="3" THEN LSET T$=C$
3790 IF B$="4" THEN RSET Y$=MKI$(C)
3800 PRINT "なまえ :";N$;" じゅうしょ  :";A$
3810 PRINT TAB(20);"でんわ :";T$;"  ねんれい :";CVI(Y$)
3820 PRINT "もっと ていせい しますか (Y/N)";
3830 B$-INKEY$:IF B$="Y" THEN PRINT:GOTO 3710
3840 IF B$<>"N" GOTO 3830
3850 PRINT:RETURN
3900 REM *** さくじょ ***
3910 PRINT N$;"さんの レコードを さくじょ しますか ?. (Y/N)";
3920 B$=INKEY$:IF B$="N" THEN PRINT:GOTO 3560
3930 IF B$<>"Y" GOTO 3920
3940 LSET N$="":LSET A$="":LSET T$="":RSET Y$=""
3950 PRINT:RETURN
```

It is a long program. Lines 3500, 3510 and 3520 are for OPEN of the file, definition of FIELD and designation of the record number, respectively. To designate the record to correct or delete with the name is more convenient, but to do so, another file must be prepared for a table showing correspondence of names and record numbers which complicates the program. Therefore, right now, record number of the objective person is searched from display of the total members. Program 7 is not made to display record numbers, but record numbers are displayed when the program is completed.

Lines 3530 through 3550 read and display the record having the designated record number. Line 3560 receives selection of whether to correct or to delete. is provided for the case that an error is made on the record number and the read record is not the objective record.

Line 3570 checks if the input is correct. If was selected, the operation jumps from Line 3580 to Line 3600. In Line 3590, operation jumps to a subroutine of correction or deletion.

Sample run 7

```
RUN
ﾚｺｰﾄﾞNO.? 1
ﾅﾏｴ:ﾔﾏﾀﾞ ﾀﾛｳ                    ﾆｭｳｼｬﾋﾞ 15-4-24
ﾃﾞﾝﾜ:03(1062-4952      ｾｲﾍﾞﾂ: ｵ
1:ｼｭｳｾｲ    2:ｻｸｼﾞｮ    3:ｷｷｮｸ ﾅｼ? 1
    1:ﾅﾏｴ    2:ﾃﾞﾝﾜﾊﾞﾝｺﾞｳ    3:ﾆｭｳｼｬﾋﾞ    4:ｾｲﾍﾞﾂ
4
ｺﾉﾚｺｰﾄﾞﾃﾞﾓﾖﾛｼｲﾃﾞｽｶ ｵ
1 ﾅﾏｴ:ﾔﾏﾀﾞ ﾀﾛｳ                    ﾆｭｳｼｬﾋﾞ 15-4-24
ﾃﾞﾝﾜ:03(1062-4952      ｾｲﾍﾞﾂ: ｵ
ｼｭｳｾｲ ｻｸｼﾞｮ ﾅｶﾐ (Y/N) N
NEXT (Y/N)? N
Ready
```

When all processings complete, the operation returns to Line 3600 and outputs the record. Then, whether or not other record would be modified is inquired by Lines 3610 to 3620, and if modification is not longer needed, Line 3640 close the file and operation terminates as soon as E is input.

Operation from Line 3700 on is a subroutine for correction. Lines 3710 through 3730 display the objective record and inquire what to correct. Lines 3740 through 3750 inquire how to change. Lines 3760 through 3790 change the record contents and Lines 3800 through 3810 display the changed results.
Lines 3820 through 3830 inquire if to correct other items and if there is no data to correct, the operation returns.

Line 3900 and on are for a deletion subroutine. Lines 3910 through 3920 checks if the record can surely be deleted, and Line 3930 deletes the contents.

Since there are many similar processing, the program has become lengthy when compared with previous programs, but the file operation is not too complicated as there is no new instruction.

5.3.4 Summary of Random Access File

Program 10 is an address catalog program that combines all the preceding programs.

Program 10

```
100 REM ***** じゅうしょろく *****
110 OPEN "R",#1,"0:じゅうしょろく"
120 FIELD #1,15 AS N$,40 AS A$,13 AS T$,2 AS Y$
130 LOR=LOF(1)
140 CLS
150 PRINT TAB(3);"*** じゅうしょろく      ***"
```

```
160 LOCATE 10,3:PRINT "1.すべて のレコードのひょうじ "
170 LOCATE 10,5:PRINT "2.とくてい のレコードのひょうじ "
180 LOCATE 10,7:PRINT "3.ついか "
190 LOCATE 10,9:PRINT "4.ていせい "
200 LOCATE 10,11:PRINT "5.さくじょ "
210 LOCATE 10,13:PRINT "6.おわり "
220 LOCATE 1,15:PRINT "なにをしますか? "
230 B$=INKEY$:B=VAL(B$):IF B<1 OR B>6 GOTO 230
235 PRINT TAB(10);CHR$(30);B$:PRINT
240 ON B GOTO 1000,3000,2000,3000,3000,250
250 CLOSE #1
260 END
1000 REM ***** ぜんぶひょうじ *****
1010 R=0
1020 CLS
1030 PRINT TAB(30);"*** じゅうしょろく ***"
1040 PRINT " NO.  なまえ ";TAB(23);"じゅうしょ ";
1045 PRINT TAB(63);" でんわ";TAB(75);"ねんれい"
1050 FOR I=1 TO 22
1060 IF R>=LOF(1) THEN I=24:GOTO 1090
1070 R=R+L:GET #1,R
1075 IF N$="                " GOTO 1060
1080 PRINT R;TAB(5);N$;TAB(20);A$;
1085 PRINT TAB(60);T$;TAB(74);CVI(Y$)
1090 NEXT I
1100 IF I>23 GOTO 1130
1110 PRINT "なにかキーを おすと つぎを ひょうじします。 ";
1120 B$=INKEY$:IF B$="" GOTO 1120 ELSE GOTO 1020
1130 PRINT
1140 PRINT "おわりです。なにかキーを おすと メニューをひょうじします。 ";
1150 B$=INKEY$:IF B$="" GOTO 1150 ELSE GOTO 140
2000 REM ***** レコードのついか *****
2010 LOR=LOR+1
2020 INPUT " なまえ ";NAMAE$
```

```
2030 INPUT "じゅうしょ";ADDRESS$
2040 INPUT "でんわ ";TEL$
2050 INPUT "ねんれい";YEAR
2060 LSET N$=NAMAE$:LSET A$=ADDRESS$
2065 LSET T$=TEL$:RSET Y$=MKI$(YEAR)
2070 GOSUB 4000
2080 PRINT "これでいいですか? (Y/N)"::GOSUB 4100
2085 PRINT TAB(20);CHR$(30);B$
2090 IF B$="N" THEN R=LOR:GOTO 3210
2100 PUT #1,LOR
2110 PRINT:PRINT "もっとかきしますか ? (Y/N)";
2120 GOSUB 4100
2130 IF B$="N" GOTO 140
2135 PRINT TAB(20);CHR$(30);B$
2140 PRINT:GOTO 2010
3000 REM ***** 1レコード よむ *****
3010 INPUT "レコード NO.";R:IF R>LOR GOTO 3015
3013 IF R<1 GOTO 3010 ELSE GOTO 3020
3015 PRINT "レコード NO.は";LOR;"までです.":GOTO 3010
3020 GET #1,R
3025 IF N$="                " GOTO 3070
3030 GOSUB 4000
3040 ON B GOTO 140,3050,140,3200,3100
3050 PRINT "なにかキーをおすとメニューがめんにもどります. "
3060 B$=INKEY$:IF B$="" GOTO 3060 ELSE GOTO 140
3070 PRINT "レコード NO.";R;"は とうろくされていません."
3080 GOTO 3050
3100 REM ***** さくじょ *****
3110 PRINT "このレコードを さくじょしますか? (Y/N)"
3120 GOSUB 4100
3130 IF B$="N" GOTO 140
3140 LSET N$="":LSET A$="":LSET T$="":RSET Y$=""
3150 PUT#1,R
3160 GOTO 140
```

```
3200 REM ***** ていせい *****
3210 PRINT TAB(5);"1:なまえ";
3220 PRINT TAB(15);"2:じゅうしょ ";
3230 PRINT TAB(25);"3:でんわ";
3240 PRINT TAB(35);"4:ねんれい";
3245 PRINT TAB(45);"5:メニューがめんにもどる。"
3250 PRINT "なにを ていせいしますか？"
3260 B$=INKEY$:B=VAL(B$):IF B<1 OR B>5 GOTO 3260
3263 IF B=5 GOTO 140
3265 PRINT TAB(15);CHR$(30);B$:PRINT
3270 INPUT "どう かえますか";C$
3280 IF B=1 THEN LSET N$=C$
3290 IF B=2 THEN LSET A$=C$
3300 IF B=3 THEN LSET T$=C$
3310 IF B=4 THEN REST Y$=MKI$(VAL(C$))
3320 GOSUB 4000
3330 PRINT "ほかの こうもくも ていせい しますか (Y/N)"
3340 GOSUB 4100
3350 IF B$="Y" GOTO 3210
3360 PUT #1,R
3370 GOTO 140
4000 REM ***** レコードの ひょうじ *****
4005 PRINT
4010 PRINT TAB(5);"なまえ :";N$;TAB(30);"じゅうしょ :";A$
4015 PRINT TAB(5);"でんわ :";T$;
4020 PRINT TAB(30);"ねんれい:";CVI(Y$)
4030 PRINT:RETURN
4100 REM ***** YかNかの キーイン *****
4110 B$=INKEY$:IF B$<>"N" AND B$<>"Y" GOTO 4110
4120 PRINT:RETURN
```

It sure is a long program. However, it is nothing but combination of all the preceding programs basically. Different points common with other programs are processings are executed in the main routine only, read operation for display, correction and deletion is placed in one place (separate places when all records are to be displayed only), and all similar routines are placed in one place as a sub-routine. These could be realized because OPEN and CLOSE are needed only once.

The first operation is the main routine in Lines 100 through 260. It basically is the same as the main routine that controls input/output to and from sequential files, Program 3. Differences are as follows:

(1) OPEN and CLOSE were executed for each subroutine in sequential files since input and output modes had to be set. In random files, however, no mode change is needed, so that the two are executed only once in the main routine.

(2) Program 3 is for registration and display only, but Program 10 includes operation of addition, correction, deletion and display of one record as well.

The following describes each step. Line 110 is for OPEN and Line 120 sets the FIELD statement. Line 130 reads the largest record number at the time of OPEN into the LOR variable. The variable is used for adding of data ot records.

The main loop starts from Line 140, and lines up to 220 draw a menu screen. Each processing is executed if the number shown in the menu screen is designated. Line 230 receives the key input and checks the number concurrently.

Line 240 instructs to go to the processing routine as input. The display routine of a specific record starting from Line 3000 is also a display routine of the correction

and deletion to be read out.

When the processing completes and 「おわり」 of 6 is designated, the file is closed by Lines 250 through 260 and the operation ends.

Lines for 1000 through 1120 are for the routine to display records of all members and the lines are used to learn the record number of a specific person also in addition to the use of seeing data of all members, since the record number must be designated for correction or deletion.

Line 1010 is for initialization of the record number. Line 1020 is for clearing of the screen and Lines 1030 and 1040 write the heading.

The FOR-NEX loop on Lines 1050 through 1090 is for display of data for one screen, to wait for key input and then to read the next data. In one screen, data for 22 lines can be displayed, deducting 2 lines for the heading and one line at the botton from the total 25 lines.

What is described in Line 1060 is that Loop counter I is made greater than usual when data end is detected so that determination of data end is possible even after coming out of the FOR-NEXT loop. This processing is completely same as Program 7.

Line 1070 updates the record number and reads the data. Line 1080 displays the data.

Line 1100 is for determination of data ends. If data has ended, the operation returns to display of the menu screen from Line 140.

Key input is waited by Lines 1110 through 1120 and the operation returns to Line 1020 where data is read in again.

Lines 2000 through 2140 are for adding of data. If data is added, the maximum record number increases and Line

2010 updates LOR. Variable LOR is updated when data is added only, and not used in other processings.

Data input is received by Lines 2020 through 2050, set to the buffer by Line 2060, and checked if the key input was correct by Lines 2070 through 2090. The reason for checking data that has been set to the buffer is that sometimes size of the input data is too large, overflowing from the field assigned to the buffer. If there is an error in the input data, instead of receiving the data again, the operation goes to the routine of correction. In the correction routine, ⌜R⌟ is used as the record number, and variable LOR that indicates the record number being used here is assigned to R . The subroutine in Line 4000 is for display of the file buffer contents. The subroutine in Line 4100 is for wait of key input and it does not accept input other than ⌜Y⌟ and ⌜N⌟.

If there is no error in the data input, the data is output (Line 2100).

Lines 2110 through 2130 inquire if there is more data to be added. If there is, the operation returns to Line 2010 and input is received. If there is no more data to be added, the operation returns to the menu screen.

Lines 3000 through 3030 are for display of data in one record, and record of the designated record number is read and displayed. This operation is for additional purpose to check the data of before modification after completing the correction or deletion.

The operation jumps to individual processing routine by Line 3040. Any processing that does not require this routine, like ⌜1: Display of all records⌟ or ⌜3: Adding of record⌟, is returned to the main loop. The operation goes to Line 3200 if for correction or Line 3100 if for

deletion. If display is the only operation required, input of any key is waited by Line 3060, and the operation returns to the menu screen.

Lines 3100 through 3160 are for deletion processing. Whether or not the data can really be deleted is checked, in the same as in Program 9 first, and the buffer is emptied and written. When this processing ends, the operation returns to the menu screen.

Lines 3200 through 3370 are for correction routine. If an input error is made in adding the record, the operation jumps to this routine also. Data to be corrected is designated with the number in Lines 3200 through 3260. Input of new data is received by Line 3270, and the new data is written in the designated data field by Lines 3280 through 3310. Carefully note Line 3310.

Line 3320 displays the corrected results, and Lines 3330 through 3350 inquire if there is other correction to be made. If there is, the operation returns to the start of this routine, and if not, data is output by Line 3360 and the operation returns to the menu screen.

Lines 4000 through 4030 are for the subroutine that displays the data buffer contents. Pay attention to the Y$ processing of Line 4020.

Lines 4100 through 4120 are for operation of checking key in, and only when key in of either ┌Y┘ or ┌N┘, the operation returns.

Input of other key is completely ignored.
The program itself is very long but since all processing is rather simple, we believe that readers have understood the program now.

At the end, features of random access files are summarized. They are repeatings of the same points as explained in the sequential file section, but in reverse expression.

Random access files have the following advantageous points:

(1) OPEN and CLOSE need not been repeated for each mode of input and output.

(2) Data can be read out in high speed in units of record regardless of the location in the disk.

(3) Correction and deletion in units of record are possible.

Random access files have the following disadvantages:

(1) Programs required for random access files are very complicated.

(2) There is no flexibility in the size of individual data item. A part of data must be truncated if its size is longer than the predetermined size.

(3) Data is in a fixed length and written in many places of the disk, and use efficiency of the disk is poor.

On the data length and record length referred to in Item (2), there are many restrictions being imposed to them, and there are the following restrictions in addition to what have been explained:

(1) Length of one record must not exceed 128 bytes (128 characters).

(2) Number of records per file must not exceed 624.

Also, there are the following two restrictions, which are general rules to be applied to DISK BASIC, not limited to random access files:

(1) Number of files to be written in one diskette is limited to 40.

(2) Number of files that can be opened simultaneously is limited to 16.

Since there are so many restrictions, readers may wonder why there are so many, but further explanation is omitted as it requires too many pages and the explanation involves very difficult subjects. The only restriction that causes inconvenience in daily operation is the limit of record length, not to exceed 128 bytes.

The third disadvantage may be difficult for readers to understand from explanations that have been given so far. In the case of sequential files, only output is made at the time of output, so that data can be written in continuously from the point where write started. In the case of random files, however, since the record length is fixed, an area of one sector (=128 bytes) is needed to record a small data item of 50 bytes (one sector is preserved as an area for one record always). If the same area is written with a sequential file, data that is equivalent to slightly over 2-record size can be written.

Also, since individual data items are in a fixed length, it often occurs that the area is used up to about half on some data items, but the area is not enough for other items, causing a necessity of abbreviating the addresses.

These are sacrifices needed for the sake of obtaining high speed and flexible processing.

Access to disks without file operation is explained in the last section.

5.4 OTHER DISK OPERATIONS

5.4.1 DSKI$

DSKI$ is a function that returns contents of certain diskette position and it resembles the PEEK function that returns the memory contents. One difference is that the PEEK function returns data of one byte and the DSKI$ function returns the diskette contents as character strings of 128 bytes. Also, the PEEK function designates the position of data to be read out by address designation, where as the DSKI$ function designates the data position with a track number and sector number in addition to a drive number.

Data is written in diskettes concentrically by magnet. Each of the concentric circle is called a track. The tracks are numbered sequentially from outside. Each track is divided into 16 sectors and one sector consists of 128 bytes. Remember the DSKF function explained in 3.2. DSKF defines empty areas of disks in units of record and one group consists of 4 sectors.

The DSKI$ function returns data of 1 sector (=128 bytes) with the track number and sector number being designated.

This function is useful to know format and structure of files stored in disks and how sequential and random files are being stored. Much knowledge and effort are required to obtain such information by actually reading disk contents.

A sample program for hexadecimal dump of a drive, track and sector designated by key in is shown in Program 11. No instruction of OPEN or CLOSE nor FIELD statement is needed to use the DSK$ function.

In the character part, a period is used for all control cordes. Explanation of this program is omitted.

DISKCOPY explained in 1.3 UTILITY also uses the DSKI$ function. Read the part to make sure understanding the function.

Program 11

```
10 INPUT "ﾄﾞﾗｲﾌﾞ ";D
20 IF D<0 OR D>1 GOTO 10
30 INPUT "ﾄﾗｯｸ ";T
40 IF T<0 OR T>39 GOTO 30
50 INPUT "ｾｸﾀ ";S
60 IF S<1 OR S>16 GOTO 50
70 A$=DSKI$(D,T,S)
80 FOR I=1 TO 16
90 B$=""
100 FOR J=1 TO 8
110 C$=MID$(A$,(I-1)*8+J,1)
120 A=ASC(C$)
130 D$=HEX$(A)+"  "
140 IF LEN(D$)=2 THEN D$="0"+D$
150 PRINT D$;
160 IF A 31 OR A=255 THEN C$="."
170 B$=B$+C$
180 NEXT J
190 PRINT "  ";B$
200 NEXT I
210 END
```

Sample run 8

```
RUN
ﾄﾞﾗｲﾌﾞ ? 0
ﾄﾗｯｸ ? 4
ｾｸﾀ ? 13
```

```
EB E0 E1 20 E0 FB 93 20   ｶﾀﾗﾀﾗｶ
20 20 20 20 20 20 20 E4          ﾄ
93 97 8E 93 E4 20 F0 E5   ｳｷｮｳﾄﾓ ﾏﾙ
E4 98 20 E6 9C 9C FD EA   ﾄｸﾆﾓﾓﾚﾛ
DE 9C 20 31 35 2D 36 2D   ﾞﾚ 15-6-
32 34 20 20 20 20 20 20   24
20 20 20 20 20 20 20 28             (
30 33 29 30 36 32 2D 34    03)062-4
39 35 32 20 00 16 00 00   952 .....
00 00 00 00 00 00 00 00   ........
00 00 00 00 00 00 00 00   ........
00 00 00 00 00 00 00 00   ........
00 00 00 00 00 00 00 00   ........
00 00 00 00 00 00 00 00   ........
00 00 00 00 00 00 00 00   ........
00 00 00 00 00 00 00 00   ........

Ready
```

5.4.2 DSKO$

Readers may think that DSKO$ is a function, but it is not a function but instruction. DSKI$ is a function to read disk contents whereas DSKO$ is an instruction that directly writes into disks. The relation is similar to that between PEEK and POKE.

It was mentioned that careless use of the POKE instruction would destroy BASIC programs. The same, or even more careful attention is needed when using DSKO$, because, in the case of POKE, what is destroyed is a program made by you and the BASIC interpreter contained in ROM is not affected. However, erroneous use of DSKO$ sometimes may destroy the DISK BASIC itself. For this reason, use of the DSKO$ instruction is not recommended unless it is absolutely necessary.

For a sample program, refer to DISKCOPY in 1.3 UTILITY .

89

6. Use Method of RAM Disk

A memory (RAM) on S1 can be used fo store programs and data as a floppy disk is used. A RAM that is used for this purpose is called a RAM disk. Since a RAM disk can be accessed in the same speed as that to a RAM, programs and data can be input/output at a fast speed.

6.1 Setting of RAM disk

Before a RAM is used as a RAM disk, it must be set to a RAM disk. Once this setting is made, it does not have to be repeated each time the power is turned on. Set a RAM disk in the following procedures.

(1) Start a utility according to the procedures described in 3.1 Starting of Utilities. When the utility menu is displayed, key in ⌜6⌟ . The following submenu of System attribute setting is displayed.

XXXX

90

(2) Key in ⌜4⌟ . The cursor will be positioned to ⌜N⌟ on the line of RAM disk. Key in ⌜Y⌟ . A message inquirying which one among 0 to 5 is displayed. Key in ⌜0⌟.

(3) Restart the DISK BASIC using a NEWON instruction. The RAM disk can now be used.

6.2 Usable instructions and functions

All instructions and functions that can be used by the DISK BASIC can be used to a RAM disk. However, the operations of

the following instructions and functions to a RAM disk are different from those to an ordinary disk.

(1) DSKINI

A DSKINI instruction releases the RAM area (except the 8K bytes of the fixed part of RAM disk) that the RAM disk has been using.

(2) DSKF function

The DSKF function returns the remained RAM pages. The remained RAM can be used as a variable or array area, character area, text area or machine language area, in addition to the use as a RAM disk. Supposing that all these are used as the RAM disk, the remained RAM pages can be regarded as the number of groups remained on the RAM disk.

6.3 Usable RAM size

(1) Fixed part

When the DISK BASIC is started, 8K bytes in the RAM is reserved as the file control cylinder area (Fixed part of RAM disk).

If this file control cylinder RAM area cannot be reserved, the DISK BASIC is started in a form of unable to use the RAM disk.

(2) Reserving of memory

(a) When a sector of unused group in the RAM disk is

is accessed (read/write), a RAM area is allocated to the group (16-sector units). If there is no RAM area to allocate, a "Disk Full" error results in.

(b) Even when a sector of a group to which no RAM area has been allocated in the RAM disk is accessed by a DSKI$ or DSKO$ statement, a RAM area is allocated to the group. This memory cannot be reset until the RAM disk is initialized by a DSKINI instruction.

91 (3) Resetting of memory

(a) When a file in a disk file is deleted by a KILL, SAVE or SAVEM instruction, the memory of the group that the file was using is reset.

(b) When a DSKINI instruction is applied to a RAM disk, all RAM areas used by the RAM disk are reset except the fixed part explained in Item (1).

6.4 Effective use of RAM disk

When a RAM disk is used, data can be input/output four to six times as fast as that of a mini-floppy disk. This function is especially useful on such operations as sorting of data recorded on a diskette or processing of large amount of data like sequential retrieval.

[Example] Use examples of RAM disk

(1) Copying of a data file to use from a mini-floppy

disk to a RAM disk.

(2) Data processing of a data file in a RAM disk

(3) Copying of a data file that has been processed into a mini-floppy disk in the reverse procedures of Item (1)

92 7. Reference material

7.1 ASCII code

ASCII code expresss a character, and each character code corresponds to a character in the interlace or non-interlace mode. Character codes are expressed in hexadecimal notation ⌜&H ⟨X value⟩⟨ Y value⟩⌟. For example, the character code for ⌜A⌟ is X=4, Y=1 and it is expressed as ⌜&H41⌟. In BASIC, a hexadecimal number can be used as it is but with ⌜&H⌟ written at the start, and to find out the value in decimal notation, key input ⌜PRINT &H41⌟ when waiting for a command. Note that the character displayed on the screen for the same character code is different depending on the screen mode (interlace mode or non-interlace mode). Also note that, when X=0 or 1, the character is not output to the screen. In the case of BASIC, since ⌜&H7F⌟ is used as the code for [DEL], ⌜ ⌟ is not output. Also, ⌜&H15⌟ is used as the [INS] code.

1. In interlace mode 2. Upper 4 bits

3. Lower 4 bits

93 1. In non-interlace mode 2. Upper 4 bits

3. Lower 4 bits

7.2 Error messages of DISK BASIC

1. Error me-sage 2. Error code 3. Remarks
4. The FOR ~ NEXT statements are not corresponding properly. (Either there are too many NEXT statements, or two or more NEXT~FOR statements are crossing.)
5. The program is not in the proper syntax (there is an illegal statement in the program).
6. The GOSUB - RETURN statements are not corresponding properly (RETURN statement only is given).
7. No data is prepared in the DATA statement to be read by the READ statement (either no DATA statement is given or a READ statement is given after reading all data).
8. The statement function is not called properly.
9. The operation result or input numeric is out of the allowable range.
10. No more memory capacity (The program is too long, the array is too large, etc.)
11. There is no line that is needed (destination of GOTO).

12. The subscript of array variable is not within the predetermined range.
13. Double definition of an array or FN function was attempted.
14. "0" was specified for the divisor.
15. Execution of a statement (DEFFN statement, INPUT statement, etc.) was tried even though the statement should not be used as a direct command.
16. The data type does not match between the left and right sides of the expression or function arguments (numeric vs. character, etc.).
17. The memory area for character variables specified by a CLEAR statement is short.
18. The character string is too long (the result of character expression is longer than 255 characters).
19. The character expression is too complicated (the nesting of parentheses is too deep).
20. Cannot continue the program execution by a CONT instruction (the pointer is destructed).
21. An attempt was made to refer to FN function not defined by a DEFFN statement.

95 1. Error message 2. Error code 3. Contents
4. Unable to restart the program execution after an error handling.
5. The error and RESUME are not corresponding (RESUME was

applied while there was no error).

6. An error having no defined message occurred.
7. A necessary parameter in the statement is not specfied.
8. The FOR ~ NEXT statements are not corresponding correctly (too many FOR statements).
9. WHILE and WEND are not correctly corresponding.
10. A File No. that is not opened is specified..
11. An attempt was made to execute an I/O instruction that did not correspond with the mode at the time of OPEN.
12. Opening of a file that had already been opened was tried.
13. An error occurred on read/write of a used device.
14. A file input statement was executed after reading all data in the file.
15. The file descriptor contains something that should not be there.
16. A direct statement was found during execution of LOAD in the ASCII form (when LOAD was applied to a data file).
17. An attempt was made to access a file that had not been opened (an attempt was made to apply an instruction that would need a file without opening the file).
18. The data form on the file is incorrect.
19. An attempt was made to use a device which was other than a disk and which was being used currently (by a command or OPEN instruction).

20. The device is not in the usable state (the interface is mounted but the connector is not connected, or the like).
21. The I/O buffer is overflowing (the file cannot be read correctly, or the like).

96 1. Error message 2. Error code 3. Contents

4. An attempt was made to write or revise a protected file.
5. A file that is not on the disk is specified.
6. Since the directly is filled, no file can be made.
7. An attempt was made to concurrently open a file that is greater than the number of FCB (File Control Block).
8. An attempt was made to extend or create a file in the state that all groups had been used (no more user area).
9. A file name existing on the same diskette is specified for a new file by a NAME command.
10. The total size in the FIELD statement is greater than the record length.
11. No meaning (not being used) on this DISK BASIC.
12. A record No. of below 1 is specified in PUT or GET, or the area up to the Record No. specified by PUT cannot be obtained on the disk.
13. The file is not structured correctly. (FAT or directory is incorrect.)

14. No diskette is mounted to the specified drive.
15. An attempt was made to write onto a diskette that had been given the write protect function.
16. The length of record on the randam file to be opened (total record length when two or more files are to be opened) is longer than the field buffer size.
17. When a sequential file is read in the "R" mode, an attempt was made to read beyond the last data (this is used by a future extended DISK BASIC but not by this DISK BASIC).
18. The written data and read data do not concur in the VERIFY ON state. (There is a possibility of the data not being written correctly.)